Examensarbete MEE06:10

Situation-dependent spontaneous mobile information service for travelers

Tobias Bertilsson
Mårten Persson

This thesis is presented as part of the Degree of Master of Science in Electrical Engineering with emphasis on Internet Systems

Blekinge Institute of Technology
2005/2006

Master of Science in Electrical Engineering with emphasis on Internet Systems
Blekinge Institute of Technology
School of Engineering
Department of Telecommunication Systems
Supervisor and Examiner: Markus Fiedler, Blekinge Institute of Technology
Supervisor: Maria Jernbäcker, Saab AB

Special Thanks

First of all we would like to thank **Maria Jernbäcker** at **Saab AB** for the opportunity to do this thesis work, and for the help with the insight to make the application more user-friendly.

Then we would like to thank **Markus Fiedler**, our supervisor at **BTH**. Thank you for the opportunity to present our ideas on the national ITS-conference in Stockholm, autumn 2005. Thank you for the discussions when we got stuck, there your endless knowledge has helped us a lot.

We also want to thank **Annelie Walter, Jonas Andersson** and **Fredrik Erlandsson**, who has taken their time to review our report, and testing the application. You have also given us a lot of mental support.

Ulf Andersson has also helped us; he provided some space on his server so that we could test the application outside its normal environment.

Thank you all, without your help this thesis wouldn't have been possible to do!

Tobias Bertilsson and *Mårten Persson*

Abstract

Situation dependent spontaneous mobile information service for travelers

This application is created to provide valid and relevant travel information to travelers on their journey. The application is proposed as a demonstration for seamless communications in the VINNOVA-sponsored project PIITSA (Personal Information Intelligent Transport systems through Seamless communications and Autonomous decisions).

The PIITSA-project is a cooperation between Blekinge Institute of Technology (BTH), Saab AB, SP Swedish National Testing and Research Institute and Swedish Road Administration region Stockholm. The project is going on under three years, and the goal is to define communication and decision functions which will make it easier for ITS-applications (Intelligent Transport Systems and Services) that have the need of supplying personally information.

This project presents a solution to the question “How do I get from point A to point B in any case”. All that the user needs is a PDA with the application installed and being able to connect to the service via a mobile network. If the user has specified a route in the application, then the user will receive up-to-date information on how to get to his destination. This information is presented on the user’s PDA. If something happens on the current route, then the user will get an automatic message sent to him presenting the problem and give him a new travel route. Thanks to this the user won’t be stuck in the traffic problem and will probably reach its destination in time.

Thesis work:
Master of Science in Electrical Engineering,
with emphasis on Internet Systems
2005/2006

Supervisors:
Markus Fiedler at the department of Telecommunication Systems,
Blekinge Institute of Technology.
Maria Jernbäcker, Saab AB

Students: Tobias Bertilsson: inb00tbe@student.bth.se
Mårten Persson: inb00mpe@student.bth.se

Table of Contents

1. INTRODUCTION ---------- 7

1.1. Involved Parties ---------- 7

1.2. Background and Motivation ---------- 7

1.3. Goal of the Thesis Work ---------- 8

1.4. Structure of the Report ---------- 8

2. DEVELOPMENT PROCEDURE ---10

2.1. Services ---------- 10

2.2. Target Groups ---------- 11

2.3. Graphical Structure Analysis ---------- 11

2.4. Information Sources ---------- 12

2.5. Different Cost of Services ---------- 12
2.5.1. Pay When Used ---------- 13
2.5.2. Purchase Application ---------- 13
2.5.3. The Combination ---------- 13

2.6. Security ---------- 13

2.7. Competitive Services ---------- 14

2.8. Other Information ---------- 14

3. DEVELOPMENT OF TRAVELPLANNER ---------- 15

3.1. Services ---------- 15
3.1.1. Search Route ---------- 15
3.1.2. Cover Route ---------- 16
3.1.3. Receive Event When Something Happens (Added Value) ---------- 16

3.2. Target Groups ---------- 16
3.2.1. Target Group 1 ---------- 17
3.2.2. Target Group 2 ---------- 17
3.2.3. Target Group 3 ---------- 17

3.3. Graphical Structure Analyze ---------- 18

3.4. Information Sources ---------- 19
3.4.1. Traffic Information ---------- 19
3.4.2. Public Transport ---------- 19
3.4.3. Road Information ---------- 19
3.4.4. Streaming Information ---------- 19

3.5. Cost in TravelPlanner ---------- 20

3.6. Security ---------- 20

3.7. Competitive Services ---------- 20
3.7.1. trafiken.nu (Stockholm) ---------- 21
3.7.2. Eniro ---------- 21
3.7.3. Stockholm Transport (SL) ---------- 22
3.7.4. Västtrafik ---------- 23
3.7.5. Mobidock ---------- 23

3.8. Positioning ---------- 24
3.8.1. Automatic Positioning ---------- 24
3.8.2. Pushed Positioning ---------- 24
3.8.3. No Positioning ---------- 25

4. SYSTEM DESIGN ---------- 26

4.1. Different Designs ---------- 26
4.1.1. Centralized Design ---------- 26
4.1.2. Decentralized Design ---------- 28
4.1.3. Hybrid Design ---------- 30

4.2. System Structure ---------- 32

4.3. Use Cases ---------- 33
4.3.1. User ---------- 33
4.3.2. Content Provider ---------- 35
4.3.3. Service Provider ---------- 36

4.4. Sequence Diagram ---------- 37
4.4.1. Login Procedure ---------- 37
4.4.2. Search Procedure ---------- 38
4.4.3. Insert Route Procedure ---------- 38
4.4.4. Update Route Procedure ---------- 39
4.4.5. Delete Route Procedure ---------- 39
4.4.6. Event Procedure ---------- 40
4.4.7. Logout Procedure ---------- 40

4.5. Database Design ---------- 41
4.5.1. Database Server Core ---------- 41
4.5.2. PDA Database ---------- 43

4.6. Class diagram ---------- 44
4.6.1. Client ---------- 44

4.6.2. Server----------45

5. RESULT----------47

5.1. Measurement of Data Traffic----------47

5.2. The Cost of the Service----------48

5.3. The Information Sources ----------49

5.4. Chosen System Design----------49

5.5. Services of the Application ----------50

6. Problems ----------52
6.1. Compact Framework ----------52
6.2. Exiting the Application----------52
6.3. Positioning ----------53
6.4. Database Connectivity----------53
6.5. Mobile Database ----------53
6.6. PDA Problem----------54
6.7. Search Query to Stockholm Transport -----54
6.8. Problem with the Event Reception----------55
6.9. Collecting Traffic Events ----------55

7. DISCUSSION ----------56

8. CONCLUSION----------58

9. FUTURE WORK ----------59

10. REFERENCES----------61

10.1. Paper References---------- 61

10.2. Digital References---------- 61

11. INDEX----------63

APPENDIX A – GLOSSARY----------66

APPENDIX B – EXPLANATION OF TERMS ----------67

APPENDIX C – SEQUENCE DIAGRAMS----------72

APPENDIX D – USER MANUAL -----81

APPENDIX E – CONFIGURATION MANUAL ----------96

List of Figures

Figure 1: The application ---------- **18**
Figure 2: Centralized design: Planning ---------- **26**
Figure 3: Centralized design: Event ---------- **27**
Figure 4: Decentralized design: Planning ---------- **29**
Figure 5: Decentralized design: Event ---------- **29**
Figure 6: Hybrid design: Planning ---------- **30**
Figure 7: Hybrid design: Event ---------- **31**
Figure 8: System structure ---------- **32**
Figure 9: Use case: User ---------- **33**
Figure 10: Use case: Content provider ---------- **35**
Figure 11: Use case: Service provider ---------- **36**
Figure 12: Database design: Server core database ---------- **41**
Figure 13: Database design: PDA database ---------- **43**
Figure 14: Class diagram: Client ---------- **44**
Figure 15: Class diagram: Server ---------- **46**

1. Introduction

1.1. Involved Parties

This project is a cooperation between Blekinge Institute of Technology (BTH) (supervisor Markus Fiedler, master's students Tobias Bertilsson and Mårten Persson), Saab AB (Maria Jernbäcker), Stockholm Transport (SL) and Swedish Road Administrations Region Stockholm (Vägverket).

1.2. Background and Motivation

An increasing number of people travel with public transports, especially in the large cities. Car travelers have to pay considerable fees to drive into the city areas, there are too few parking lots and people are getting more involved in taking better care of the environment. At the same time people have less time, are more stressed and in need of new good technologies. They can't afford to be late to meetings etc. For these people it is out of the question to get stuck in a traffic jam in the subway or on the bus. There is one more sequence, in order to avoid getting stuck; traveling individuals need up-to-date traffic information that correlate with their routes.

Thus the PIITSA-group found a need for creating a seamless communication technology, to prevent that the users will loose their connectivity. They needed an application that could demonstrate this technology. So the background to this thesis work is that they needed a demonstrator to help demonstrate their new technique to provide seamless communication. The application should be a mobile solution to a travel planner. This application should not be depending on SMS, WAP or any other older methods to send or receive data. It shouldn't be depending on a certain kind of network, instead it should use the best network that is available at the moment, preferably WLAN or GPRS/UMTS.

A user scenario could be that Pontus wants to travel between Älvsjö and Kungsträdgården. Pontus types in Älvsjö and Kungsträdgården in his travel planner and receives the travel information which is current for him. Then an accident occurs on the subway, which was included in his travel route. The user automatically receives a message on his PDA that informs him about the accident and presents a new travel route. Now Pontus won't be late to his meeting in Kungsträdgården because of the accident.

The application should be created by Tobias Bertilsson and Mårten Persson at BTH, and should be able to be installed on the users PDA's. The theoretical part

of the creation of the application was made in the course Mobile Services (ETD013) at BTH. The result of this study was then refined and implemented; this part of the project is their master's thesis. The application should gather information about the travel route that the user wants to travel and present it for the user. The application doesn't depend on the network; it should just receive the information that would be streamed out from the system and do necessary operations with the information.

1.3. Goal of the Thesis Work

The goal of this thesis work was to create an application for mobile units, especially PDAs, so that the users should be able to search for routes from point A to point B. They should also be able to cover a certain route, for example subway line red. But the most interesting feature is that they will receive a message when something happens either on the route that they have searched for, or on any of their covered routes. Here lies the very challenge in the thesis work.

1.4. Structure of the Report

Chapter 2 will describe the theoretical background that is needed for creating an application. It will include the most vital things such as finding the services, target groups etc.

In chapter 3 the theories will be applied on the application that is created in this report. Its services and existing competitive services etc. regarding the application will be described.

The system structure of the application will be covered in chapter 4. Here different types of system design will be discussed. Also database design and sequence diagram will be mentioned and discussed in this chapter.

In chapter 5 the focus will be on the result, describing the system design to be used, estimations of data traffic cost etc.

Chapter 6 denotes some problems that occurred during implementation of the application and discuss possible ways to solve them.

Chapter 7 contains the discussion.

In chapter 8 the work is summarized.

In chapter 9 some future work is pointed out, such as aspects that may improve the application or things that need to be done to have a scalable application in the future.

Chapter 10 contains all the references that have been used, both literary and digital ones.

In chapter 11 an index for the report will be found.

Appendix A presents a short glossary of the report.

Appendix B explains terms and definitions used in the report.

In appendix C all the sequence diagrams are placed, giving overview of how the application works.

Appendix D contains the user manual. Here will the user learn how to use the application and its services.

In appendix E all necessary information about how to configure the application will be found.

2. Development Procedure

The first thing that has to be done before the design and the implementation of the application is the feasibility study (pre-study). In this study the developers should look more closely on how the application's structure should be built up, how they could make it user friendly, and they should try to find a target group that the application is to be created for. Another reason for doing the feasibility study is to localize all the needs in the system, i.e. which prerequisites are there, which structure the application should have and what kind of services the application should provide, whether there is a need of services from third parties, which costs that can be identified, etc.

If there is a need to collect information from some content providers, this need and the nature/kind of information has to be identified here.

Another very important issue to study addresses security. What kind of security should the application have and why, what are the threats?

If the feasibility study is done correctly, then the designer has a road map on how to create the application. The major issues are already decided before the implementation starts. Of course, changes can be made on the way, but they are easier to perform since the designer has good documentation behind him.

2.1. Services

The next step in the planning phase is to try to define which services the application will provide. This is that will attract the users to the application, so this part needs some serious thinking.

If there are any services that are unique, then these services should be pointed out. These particular services might make possible investors invest in the product, or convince the leading personal that your project is something that they should go for.

There should also be some information about how the users will be able to use each services, i.e. where in the application are the services connected, and if the users has to have some prerequisites of which kind.

2.2. Target Groups

When working out a structure for the application, it is important to try to keep the user in the center. The easiest way of doing this is to try to find out the target groups for the application.

Here, a target group means a group of users which has some common prerequisite [9].

If the target groups can't be figured out, then an overwhelming application could be created, which will be worthless since it was made for the wrong target group. Things that should be taking in concern are:

- Age group
- Technology skill
- Education
- Financial status
- Gender
- Environmental

All these are important, which one of them that are the most important depends on the application. It can be different from case to case, but age group and technology skill group are very often among the most important when creating an application. When the target groups for the application have been defined, the challenge in trying to create an application that suits as many of these groups as possible. This is very important, if this is made in a good way it will mean that there are many users that will have need for the application. If the application will be commercial will this result in more income, and most developers and creators are looking for this.

2.3. Graphical Structure Analysis

One major reason for performing a graphical structure analysis is to find a functional structure for the application. A functional structure means a structure that seems logical for the user. If the application doesn't have a functional structure, then the users won't feel comfortable using it.

The graphical structure analysis also makes the work much easier for the developers when the implementation has started. If they have a well performed analyze to work against (preferably with some simple sketches), it will be much harder for them to make any major mistakes and their job will be performed much

faster. This saving of time and effort can be invested in creating some extra features for the users.

In the analysis phase, the different important areas of the application should be defined, i.e. which one is the most important one for the user, and which one will give the developer most job to create the functionality for, are there any static or dynamic areas?

When developing a graphical application, the colors are also very important, not only to make the application look nice, but also to try to find colors that provide good contrast to each other. This is to make it easier for color-blind users or users with bad sight to use the application.

2.4. Information Sources

The next step is to try to decide if there is a need of some information from any content providers or a third party. The kind of information the application needs should be defined as precisely as possible. The next step is to find someone who could provide this information. If this isn't possible, the developers have to decide if it is possible to create this information by themselves. If it isn't, then it might be worth thinking if this service is necessary.

One example on this could be that you are developing a weather application that will display weather information. This applications need to collect this weather information from some content providers. One possible content provider could be SMHI in this case.

2.5. Different Cost of Services

Another important phase in the development procedure is to try to find the eventual costs in the application. Is there anything that has to be bought or hired from a third party?

These costs should be defined as good as possible. Who is providing the services, and approximately how much will it cost to use them? These factors are relevant for the end-users too; since it will play a major role in how much they have to pay for using the service. There are three different types of payment that the creator of an application could choose between. Either will the user pay when the user uses the application, or pay a one time fee when buying the application. The third solution is a combination between the other two choices.

2.5.1. Pay When Used

The user will be charged a small fee when the user uses a service. This fee will be based on which service the user uses, how long the user will use it, how many services are offered and how much information that will be sent over the network. This type of payment for services is quite typical today.

2.5.2. Purchase Application

The other solution is that the fees are included into the price that the user will pay for the application (if it should cost anything). Then the user won't be charged when using the services. This is the easiest way for the developer, as the developer doesn't have to create a billing service. The developer knows that he will get paid for his product as soon as the users buy the application.

2.5.3. The Combination

This technique is probably the most used of them all, which is a reliable source of income. First the users pay for the application when they are buying it. Then they have to pay for the service every time they want to use it. In this way, the distributor can put a low price on the application, and a quite low fee on using the service. This is possible, since they know that they will receive enough payment anyway.

2.6. Security

In these days, security is a big issue to think of when developing an application. Which data that needs to be encrypted and which data could be left as it is, needs to be defined. If the application is stationed in at least two different places, then the data that are necessary to be send between the different nodes also need to be defined. To create a secure application, the amount of data traffic should be as low as possible.

With personal information about the users, such as social security number, addresses and so on, there is a need of even more security. This information isn't allowed to be stored on the same way as all other data. In Sweden, there is a specific law that explains how to handle this kind of information (PUL).

2.7. Competitive Services

This is a very important phase in the development procedure. If this isn't done in the right way, the developer could be standing there with a product that there are no more market for. The different services that are out there today are examined in this phase. What these services can do and why the user should use the particular application that the developer is developing instead. This is some questions that the developer needs to ask himself.

Every competitive service should be examined and documented. What are the pros and cons for the service in question? Which is the most valuable service that the competitors are providing? Is there anything that can be improved?

These are some question that could lead to developing an application that the user prefers instead of the competing ones, and so the application/service under development finds it market.

2.8. Other Information

In the developing procedure it is important to try to pinpoint as many services as possible. This may feel like doing unnecessary work, but it will be worth it when it comes to the developing and implementing phase. The time and effort that are saved here can be used later on to create more features for the users.

But this will only work as long as the findings are well documented. It will save time, effort and money, and the amount of mistakes will be smaller. Given a precise developing document, the consequence of any possible mistakes will probably be quite small.

3. Development of TravelPlanner

Much of the theoretical pre-studies were carried out in the course ETD013 Mobile Services at Blekinge Institute of Technology in spring 2005. The resulting service design was then refined and developed further so that it could be applied in this thesis work. The reason for doing this was that new and better solutions were discovered, which we preferred to use instead of the old solutions.

3.1. Services

Three major services can be defined in the application. They are “Search Route”, “Cover Route” and “Receive Event”. The first two can be performed by the user anytime as long as the user is connected to the service. The last one occurs when the user has either made a search for a route between point A and point B or if the user has at least one covered route in the database. If something happens on either a covered route or on a searched route, then the user will receive an event.

3.1.1. Search Route

The search route service is quite easy to understand. The user wants to travel from one place to another, inputs the name of the departure station, and then the name of the arrival station. Then the user has to decide which type of transportation to use (train, subway and/or bus). If there are any roads/routes that the user doesn’t want to travel on, then their name should be typed in here and they will be excluded in the search. This service is provided by a third party.

When the user clicks on the search button, the search result is displayed on the PDA. Now can the user decide to accept the search result or ask for a new search. If the user decides to accept the result, then the route is inserted into the user profile.

Prerequisites: The user has to know the exact names on the stations.

3.1.2. Cover Route

This is a powerful service that the application offers. Here the user can create covered routes that will be inserted into his profile in the database. The user can name the covered routes, decide which type of transportation (bus, train, subway or roads), which weekdays (or a specific date) and which time that the application should receive information about these covered routes.

Prerequisites: The user needs to know the name of the route that he wants to get information about.

3.1.3. Receive Event When Something Happens (Added Value)

This is the most important service that the application provides. It is this service that will make the users willing to use the application. If something happens on the searched route or on a covered route, then the user will automatically receive a message that explains what just has happened on the route. Now the user can search for a new route just by clicking the “alternate route” button. This will send a new query to Stockholm Transport [13] that will create a new route around the problem. This new route will be presented for the user and can be stored into the user’s profile.

Prerequisites: The user has to have at least one active covered route in its profile.

3.2. Target Groups

The great target group for the TravelPlanner application is the same who also are the target for mobile services on PDA. They are men and women mostly in the age between 18 and 40, probably quite successful in their life; they want to be up to date regarding modern technology and related services.

To make it easier to understand that group, it is divided into three subgroups.

3.2.1. Target Group 1

The first group are the “typically city travelers”. The members in this group spend most of their time in the city. They use public transportation several times per day. They use all kinds of traveling, not only bus or subway.

Demand: Collect as many different types of transport as possible in the application.

3.2.2. Target Group 2

The second group is the “regular users for this kind of new technologies”. They are well updated and try every new technology that is available. They don’t hesitate to pay for the services that they are using.

Demand: Have something new and unique that is really powerful to make this group start using the application.

3.2.3. Target Group 3

The last group contains the beginners. This group is almost afraid of new technologies. If the technology is too complex they will never use it, because they think that something will get broken. But in the same time they are curious. They want to try new things.

Demand: Make the application easy to use, but still robust.

3.2.4. Solution

To find a structure that fits all these three target groups, the application is constructed such that it will have two main parts, menu and main window. The application collects data for the four major types of urban traveling, bus, subway, train and roads. This is the most interesting feature for target group 1, thanks to this service they will have information about all public transport nearby. Target group number 2 will enjoy the application thanks to its unique service, the event handler. When something happens the users concerned by this will automatically get information about this event. This gives the user the possibility to easily make a new search in order not to get stuck in the queue or whatever the event warned for. This application is easy to use, and the structure is built in a logical way so that the target group 3 won’t be afraid of using it. But at the same time, it is very

robust, i.e. nothing critical will happen if the users makes a mistake. The users can undo most of the actions.

3.3. Graphical Structure Analysis

To find a functional structure over the design we looked more closely on the different elements which together create the base of the application. These elements are the menus and the main window.

The most important element is the menus. They create a structure over the application; with the help of the menus, the user will be able to click his way through the application. The structure is heavily dependent on how well the menus are working. Shouldn't the menus work in the way the users expect them to, will this give the users the negative impression of that the application is confusing. It is through the structure that the user gets a mental overlook (how things are connected) of the application design and if this isn't positive, then the user won't continue to use the application.

The application should look as follows, cf. figure 1, It has a menu field (arrow 2, 3 and 4 in the picture below) which will always be available. Then it has a main area (arrow 1) in which all information will be displayed. This area is dynamic, and will change depending on what the users are doing in the application. This makes it easy to get an overview of the application.

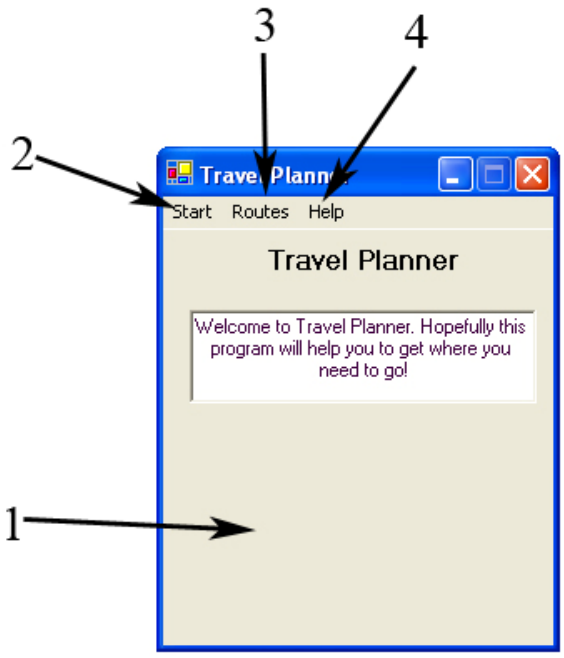

Figure 1: The application

When developing a graphical application it is important to use colors with high contrast. The application should contain three colors, white, black and grey. This will make the text easier to read.

3.4. Information Sources

The following subsections describe different types of information needed by the application.

3.4.1. Traffic Information

Information if something happened which will delay the travel. These things could be roadblocks, e.g. when the Swedish Road Administration (VV) [14] has to repair the road, but also accidents or other problems. One source of this information is VV but also Trafik Stockholm [11]. All these information are stored on trafiken.nu [12].

3.4.2. Public Transport

This is information about busses, commuter trains, subways and other ways to travel with public transport. Here, the application will get information about when the transport will depart and arrive, from where the transport will go from and where it will go to, as well as where the bus-stops and the subway stations are located. This information is primary received from SL [13] and information about delays etc. from trafiken.nu [12].

3.4.3. Road Information

This is information about the best way to travel on the roads with i.e. a car or a motorcycle. This also includes information about which condition the roads are in, e.g. if it is ice or snow or if the road conditions are good. This information comes from trafiken.nu [12].

3.4.4. Streaming Information

To stream out the information DAB could be used. In this case all the information is streamed out to all the users. So for example if something happens on a train all users get the information, and it is up to the application to determine whether the information is relevant for the user or not.

3.5. Cost in TravelPlanner

The first type of cost that was identified is the data traffic cost between the PDA and the server. It is the user that will be charged for this traffic depending on what kind of mobile subscription the user has.

The second type of cost is the cost for collecting the traffic information.
At the moment there are four different services that might be used:

www.sl.se	Stockholms Transport [13]
www.vv.se	Swedish Road Administration [14]
www.trafikstockholm.com	Trafik Stockholm [11]
www.trafiken.nu	trafiken.nu [12]

This information is for free, its “official information” and is paid by taxes.
In this thesis work, only SL and trafiken.nu will be used.

3.6. Security

The only information that needs to be encrypted is the information that concerns a user’s position, i.e. “Pontus’s location is N 53 degrees 44’77’’ E 20 degrees 29’32’’”. This is the only information that could be of interest for other people, which could use this in a undesirable way.

All the other information that will be sent concerns only travel routes between two places. It won’t be any harm if someone finds out which way between two places is the best.

This will only work as long as the user can’t be identified. The information in the user’s profile contains only information about the whole route and not about parts of the route. So if the user has a route which contains a switch between subway Red and subway Green, then only the information Red and Green will be stored. The information of where the user will switch subway will not be stored.

3.7. Competitive Services

There are five competitive services for our application. Not all of them are for mobile devices, so they could hardly be called competitive services. None of these five services had any real-time information that is sent to the users. The user’s gets the information then they ask for it, if something happens along the way they won’t receive any information about it if they don’t explicitly ask for it.

3.7.1. trafiken.nu (Stockholm)

http://www.trafiken.nu/
Trafiken.nu is a service that is available both on a stationary/laptop PC and on a mobile unit. When the service is used on a mobile unit, WAP technology is used. Trafiken.nu offers information when the user wants to travel by public means of transportations. Trafiken.nu doesn't only provide information about the best route between point A and point B; they also count in traffic information when they calculate a route. Trafiken.nu receives their information from among others SL, VV and Trafik Stockholm. Then they refine the received information and offer it to everyone on a public web- or WAP site.

Pros:

- Provides good information about public transport and walking distance.
- Displays how long time the route will take and its subroutes.
- It is possible to look into live camera photos, get information about road messages, average speed of the route, parking places and weather conditions.
- Works with every technology that has the possibility to use WAP or WWW.

Cons:

- Doesn't include traveling with cars when they calculate their routes.
- The user doesn't get any real-time information automatically if something happens that will affect its route.
- The only time the traffic condition (messages) is used is when the route is calculated in the beginning.

3.7.2. Eniro

http://www.eniro.se/
Eniro provides a service for car travelers; travelers who want to travel from point A to point B and only use their car to get there. Eniro is available both on the Internet via a PC and through WAP via a mobile unit, i.e. a PDA or a mobile phone.

Pros:

- Displays good information for traveling with cars, includes all road routes.
- Works for everyone who has the possibility to use WAP or Internet.

- It's possible to see a roadmap for the starting point, the end point and for the whole trip.

Cons:

- Is only working with car traveling.
- Has no traffic information.
- The mobile part is only working for those who have a Telia or Vodafone Live subscription.

3.7.3. Stockholm Transport (SL)

http://www.sl.se/
This service provides good information about public transport. It's from here that trafiken.nu gets its information about public transport in the Stockholm area. This service is one of the few that includes traffic information when they calculate the routes. Here it is possible to browse through traffic deviations that occurred earlier but still are valid. But also here, it isn't possible to include traveling with a car in the route.

Pros:

- The user can get information about traffic deviations, busses and commuter train.
- Provides information about time, stations etc.
- It's easy to find information about a certain station, even if the user doesn't know the exact name of the station.

Cons:

- The user can only do a search between two stations. If the user specifies an address that isn't a station, the service guess that the user meant the station whose name is most like the specified address.
- Provides only information about public transport.
- Real-time information is only delivered upon request.

3.7.4. Västtrafik

http://www.vasttrafik.se/
Provides services like: road deviations and next journey (from a specific station).

Pros:

- It's easy to use if the user only wants to get information when public transport is departing from a specific station.
- Provides information about traffic messages, road conditions and road works.

Cons:

- Is only available for public transport.
- Doesn't offer any automatic update to get new traffic information.
- Only in Gothenburg, Skara, Fyrbodal and Sjuhärad area.

3.7.5. Mobidock

http://www.mobidock.com/
Mobidock is a service that works both on a PC and on a mobile unit. It's like a portal, which the user must register to get access to. In this portal are numerous services available, one of them is a traffic information service.

Pros:

- The service is very detailed, e.g. provides several roadmaps.
- Provides very many different services, not only in traffic information area.

Cons:

- The user has to create a profile to use the service.
- The portal has a lot services that isn't necessary in the traffic information area, e.g. blogging and community.
- It could be complicated to use the portal, as it does not provide a user-friendly structure.
- The traffic part is only working for three cities, Stockholm, Gothenburg and Malmö.
- The portal provides different services for different cities.

3.8. Positioning

To be able to use positioning there are two types of positioning to focus on. They are network-based positioning and GPS-based positioning. In network-based the application finds the location by calling a service and in GPS-based uses the application the GPS hardware to send the position.

In this thesis work has three different methods regarding positioning has been investigated, namely automatic positioning, pushed positioning and no positioning.

3.8.1. Automatic Positioning

In this scenario, the user doesn't have to do anything. The information about his position is gathered automatically. Everything is happening in the background. The user only types in his destination and sends his query to the system. Then the application will automatically gather the information about the user's position. Then the travel plan is calculated and sent to the users PDA, where it is stored.

Pros:

- The easiest way for the user.
- The information about his position is always up to date.

Cons:

- Might cost some money to get the position.
- Take longer time since everything is done after that the user has "typed send request".
- This is probably the hardest scenario to implement.

.

3.8.2. Pushed Positioning

This is happening either when the user is starting his application or when making a choice in the application that sends out a signal telling the system that the user wants to be positioned. The information of where the user is at the moment is sent to the PDA where it is stored. When doing a query about the travel plan, the user is only telling the application the travel goal since the departure position is gathered from the information that was previously queried. After this information has been sent away, the travel plan is calculated and sent to the PDA which will store it and present the information.

Pros:

- Should not be that hard to implement if the positioning works as supposed.
- The user doesn't need to know the name of the nearest departure station.
- Rather easy for the user.
- Fast when the query is send, because the information was gathered earlier.

Cons:

- The user needs to do two things: querying for the position and create a search route query.
- The position that is stored might not be his or hers current position; this is the major con for this method.

3.8.3. No Positioning

The user tells the application where the departure station is, and then tells the application where the arrival station is.

With this method, no positioning is needed. All the needed information is typed in on the PDA's screen by the user. After this the information is sent to the server which forwards the question to the information system (SL, VV and trafiken.nu). Then his travel plan is calculated and sent to his PDA and stored (this is if a client based system should be used).

Pros:

- Easy to implement.
- Cheap, no need to pay any possible charges for the positioning.
- Fast, no positioning is needed.

Cons:

- The user needs to type in more information.
- The user needs to know where his current position is.

In this thesis work the "no positioning" method was used. The reason for this was the lack of time.

4. System design

The system design chapter explains different types of designs that were thought out for TravelPlanner. It also includes use cases, sequence diagrams, class diagrams and database diagrams explaining how TravelPlanner is built up.

4.1. Different Designs

Three possible design solutions are investigated: the centralized solution, the decentralized solution and a hybrid of the two of them. All of them have pros and cons. These will be discussed in this chapter.

4.1.1. Centralized Design

In the centralized solution, most of the intelligence is placed at the server. The user would just be a client, which sends queries to the server. The server would then process the queries and send the result back to the user.

The dataflow, in figure 2, shows when a user sends a request. This is done by some kind of input that the user provides. This can be the address in the search function, or pressing a button that generates a request. This goes from the user to the server (1). The server stores the query in the database (2) (3) and sends the request to the guiding system (4). The guiding system sends the route information back to the server (5), which sends the information back to the correct user (6). This information could be a webpage.

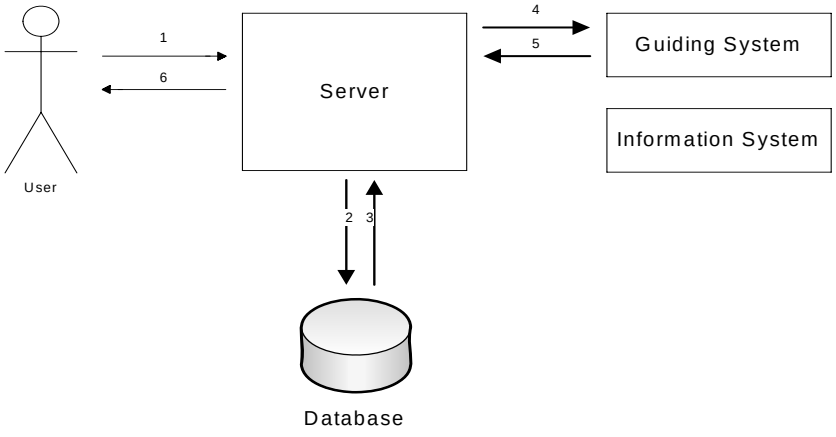

Figure 2: Centralized design: Planning

When an event occurs, where will be some minor traffic flow changes. This is shown in figure 3. The server receives an event from the information system (1). The server performs a relevancy control in the database to see which users it concerns (2) (3). If it doesn't concern any user nothing will happen. But if it concern one or more users, the server updates the database and creates a new route request and processes it to the guiding system (4). The guiding system sends back the new route for every request to the server (5) and the server sends the information to each concerned user (6).

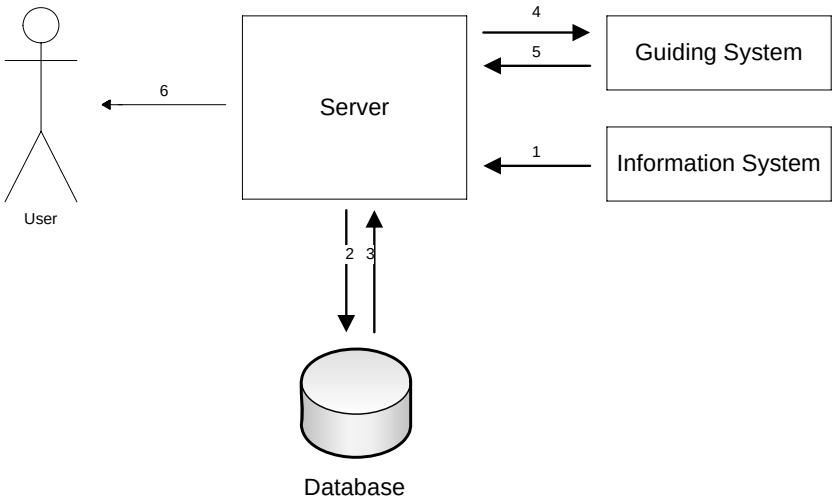

Figure 3: Centralized design: Event

The amount of data being sent from the user to the server is in this case small since all the information that been send is only the request. On the opposite direction, from the server to the user, the amount of data can be larger, i.e. a whole webpage may be transferred. The traffic flow between the server, the servers' database, the information system and the guiding system is much higher since all information in the system passes through here.

In this design, the user application could be as simple as a web client, the user could log into the system with help of an ordinary web application, i.e. Internet Explorer or similarly applications.

This design is probably the simplest to implement, since the intelligence is placed only at the server. The downside with this solution is for example that there will be a lot of data that has to be sent, especially if the web site is fairly complex. Not only the traffic information, but also the webpage will be sent as data to the user. Another problem is that the webpage needs to be adapted to be viewable on a mobile device. This could be very costly for the user, depending on the subscription, especially now when the data cost with mobile solutions (GPRS/UMTS) is still rather expensive. There is yet another problem; especially

when GPRS is used, response times in case of large amount of data might be quite large!

But the biggest obstacle with this design is that the project is locked to this design. If it needs to be changed it or upgraded it in the future, most of the application will need to be rebuilt. This means that the centralized design isn't scalable. Another problem with scalability; the server might need to deal with considerable load i.e. it might become a bottleneck, a single point of failure etc. The privacy in this solution isn't very good, since all information about the users is stored on the central server. All the information stored on the server needs to be encrypted. Storing data on one central unit also increases the vulnerability regarding attacks and system failures so that the whole information might disappear.

Depending on what type of application is chosen for the user side, the amount of data could be reduced. For example if an application written just for this type of presentation is used, then there is no need to send a whole website.

4.1.2. Decentralized Design

With this design all intelligence is placed on the PDA, which runs the application that does all the work. The server doesn't exist at all. The PDA has its own database there it stores the information about the routes, and the incoming traffic information.

With this design, the PDA is in direct contact with the content providers. There is no server that works like a gateway between the users and content providers. If the content providers send out traffic information, it will go directly to all the users. Then they will examine the information to see if it concerns them.

In figure 4 below, the dataflow can be seen. The user sends a route request to the guiding system (1). The guiding system processes the request and sends it back to the user (2). The application presents the answer of the request and stores it locally for future use (3), such as relevancy check when an event occurs.

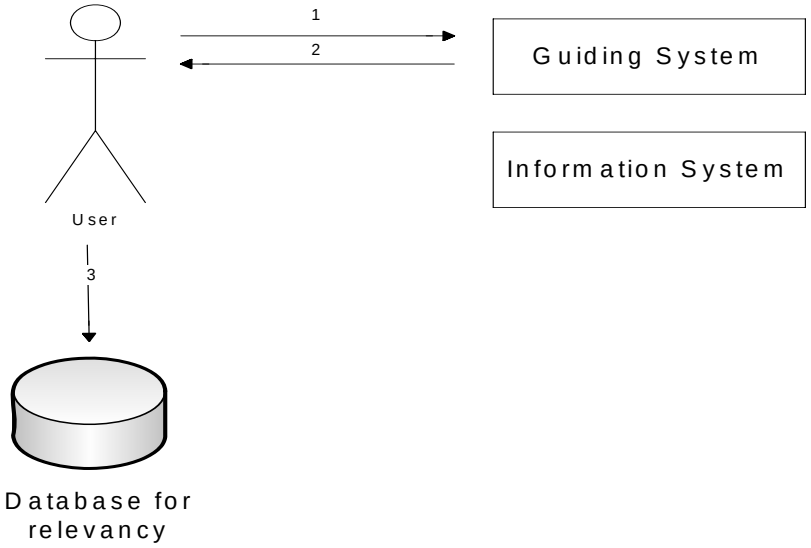

Figure 4: Decentralized design: Planning

In figure 5, it is shown what happens when an event occurs. The information system sends out all events that occur to all users (1). The application receives the event and performs a relevancy control against the local database (2) (3). If the event doesn't concern the user, nothing will happen, but if it is relevant for the user, the application displays the events, creates a new request and sends it to the guiding system (4). The guiding system then sends a new route to the user (5). The application displays the new route information and stores it in the database (6).

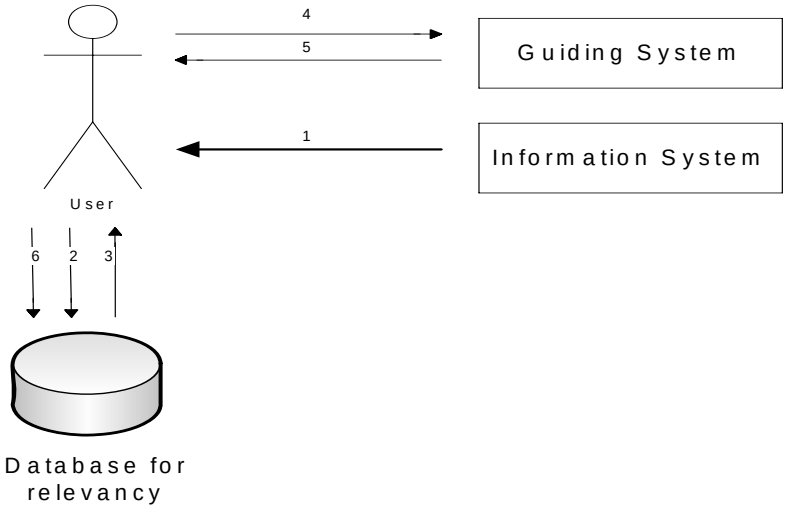

Figure 5: Decentralized design: Event

Also with this design, there will be much data that has to be sent to/from the PDA. This solution will have even more data traffic than the centralized solution,

because with this solution the traffic information will be broadcasted to all users, no matter whether the information is relevant to the users or not. This data could be sent using for example DAB. Another important issue to consider is the traffic cost. Since this is usually paid by the user, then it is important to minimize the amount of traffic sent.

The decentralized solution has the same problem with upgrades, in the future, as the centralized solution. If there are any interests to change the design structure in the future, then very much of the application has to be rebuilt.

The major obstacle with decentralized solution is that if the whole application (and all intelligence) is placed in the PDA, the battery time might be critically low. So low that the users has to consider if it is worth to have the application up and running, which is the idea with this kind of applications.

4.1.3. Hybrid Design

In hybrid design the application is divided such that it is placed both in the PDA and the server. The server will work as a gateway between the users and the content providers. This will make the communication between our application and the content providers much smoother. Now the content providers only have to send their traffic information to one place. Then it is up to the server to forward the information merely to concerned users.

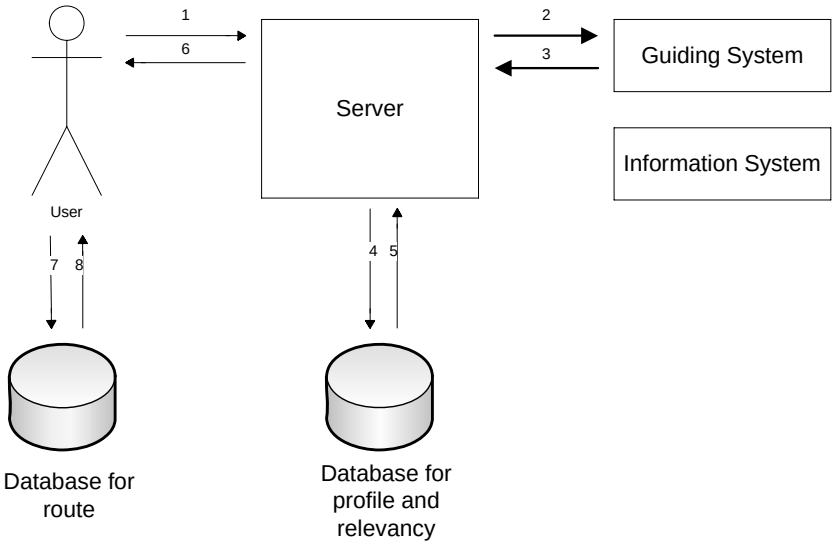

Figure 6: Hybrid design: Planning

In figure 6 the dataflow in our hybrid solution can be seen. It starts with the user making a request in the PDA. The application sends the request to the server (1).

The server sends the request further to the guiding system (2). The guiding system processes the request and sends an answer back to the server (3). The server stores necessary information in the database for future use (4) (5). The server sends the answer from the guiding system further to the application in the PDA (6). The application stores the information about the route in the local database (7) and presents the information to the user (8).

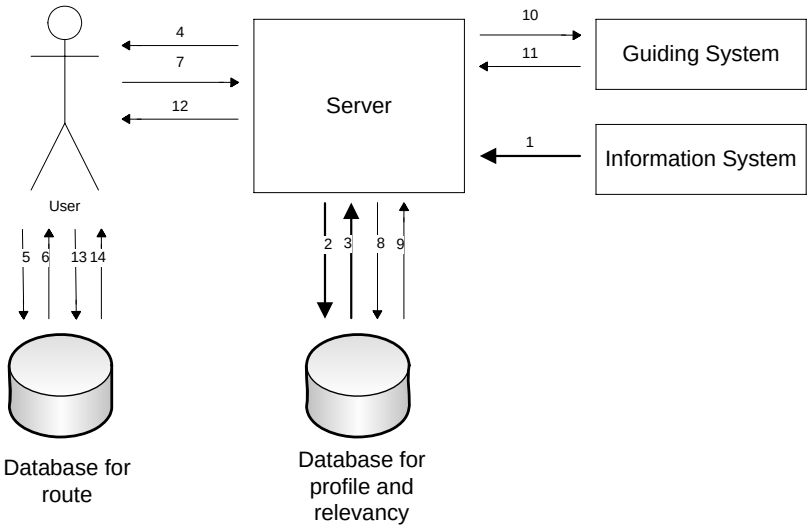

Figure 7: Hybrid design: Event

The dataflow when an event occurs is shown in figure 7. The information system sends all events to the server (1). The server performs a relevancy control in the database to see if it concerns any user (2) (3). If it does, it will forward the event to the user/users PDA application (4). The application stores the event in the database (5) (6), presents it for the user and processes a new request out of the event and sends it to the server (7). The server updates the database (8) (9). The server forwards the request to the guiding system (10). The guiding system processes the users request once again and returns the result to the server (11). The server forwards the new route to the PDA (12). The application stores the route in its local database and presents the new route to the user (13) (14).

The hybrid solution is the most intelligent solution of them all. With this solution, all parts of the system won't be loaded that hard with work. The PDA does its thing and doesn't care about what the server does. The server's only concern is to receive information and check in its database to see where to send it. If the system grows, scalability is an issue, since there will be much traffic from the server. But if some kind of broadcasting is used, e.g. DAB instead of selective streaming, then our system will be more secure for the future and doesn't have the same scalability problems as the variant described before.

The data load will be much easier with this solution compared to the previously discussed solutions. Between PDA and server, only information that concerns the user will be sent. The traffic information torrent will be much smaller too, since there only has to be an exchange of information between the server and the content providers.

The main advantage with this solution is that it is relatively upgrade-secure for the future. This solution is created so that it will be easy to put the intelligence anywhere.

This solution will be designed so that some information is stored on the server and some on the PDA. On the server side, information such as preferred bus line, subway line or specific road will be stored. This information will be used for a first filtering to see which user the information concerns. By just storing this minimal information about the user and its routes, the privacy will be kept high. On the PDA, all user-relevant information will be stored, i.e. particular route information.

4.2. System Structure

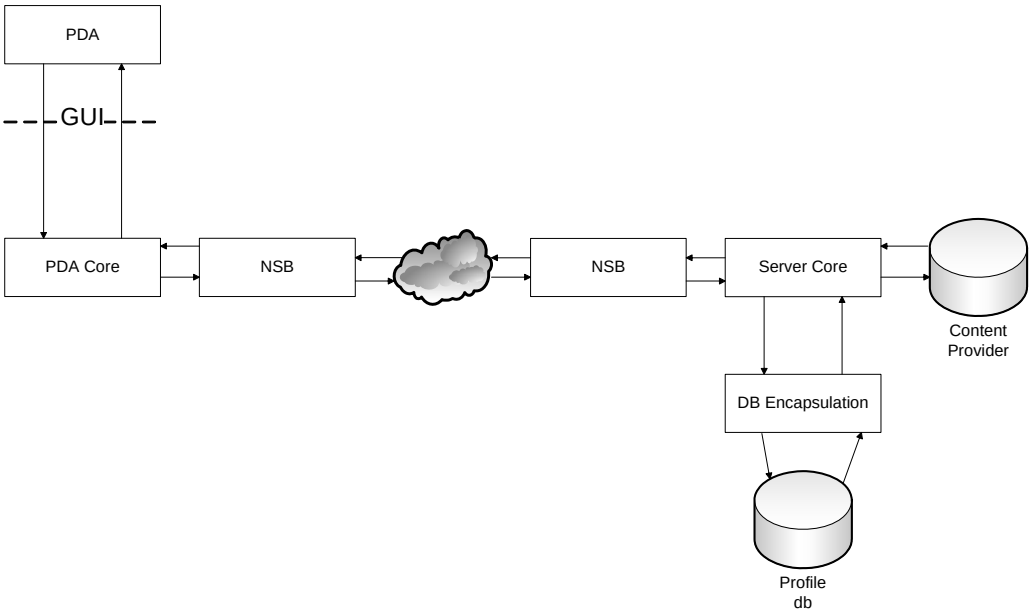

Figure 8: System structure

In figure 8 the system design is presented. There is a graphical interface (PDA) which connects to the PDA Core through a GUI. The PDA core uses the NSB to connect to the server though the net. The Server core is connected to the database through a DB Encapsulation but also has the options to connect to content

providers. The design that was chosen for this project is a hybrid. There is intelligence both at the server and on the PDA.

With the PDA the users will be able to get automatically generated information about problems with their routes. They will also be able to get new routes when their old ones would not be correct any more, which could happen for a number of reasons (accidents, delays, and strikes).

A future goal is that, when users perform searches for routes, they don't have to type in information about their starting position is. This information should be generated automatically with the help of MPS, GPS or Galileo, as discussed in section 3.8. But for now, the users will have to type in their starting position manually.

This demands that the user has some knowledge of where the departing station is, which means that the user can't be a totally stranger to the city.

4.3. Use Cases

4.3.1. User

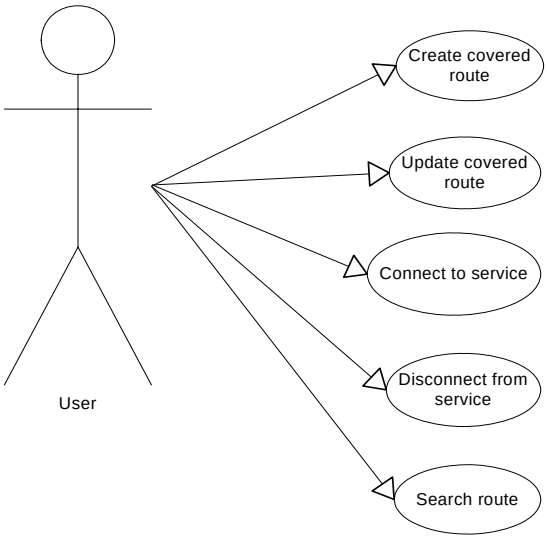

Figure 9: Use case: User

Figure 9 summarizes the options that the user has when using the application. Another way to explain this could be "what the user notices when using the application". Below, these options are explained in detail..

Create and update covered route will include these functions
When the user chose to create, update or delete a covered route, the user has to input some data into the system. These data will be stored in the database on the server. This will be done to protect the user's privacy. Here follows the information that will be stored for every profile:

- Private name on covered route
- Transportation type
- Line
- Day of validity
- Time of validity

"Private name on covered route" means that the user should have the possibility to choose a specific name for the covered route, so it will be easier for the user to recognize the route. The user could for example cover the route between his home and his work and call it "work". This name doesn't have to be unique since it is only displayed for the user that has created the covered route. All routes have a unique id, but this id is never displayed for the user, it is only used in the database.

"Transportation type" means which kind of vessels the user prefers to travel with, for example train, subway, bus or car.

"Line" means that the user specifies the line, i.e. number of the bus line, subway line etc.

"Day of validity" means that the user specifies which day the route valid. For example if the user travels each Monday, Tuesday and Wednesday, then the user should select these days.

"Time of validity" means that the user specifies which time the route valid. For example, if the user usually goes to work between 7 and 9 AM and goes home from work between 4 and 6 PM, these times could be inserted into the profile.

Connect to service
This will be the first activity the user has to perform. It is also possible to make this automatically in the future. Then the user starts the application, the unique username and password for the user that is stored on the PDA will be sent to the server. Then the user's profile will automatically be retrieved from the profile database on the server. Actions that will happen are "connect to server" and "retrieve profile".

Disconnect from service
This will be the last activity the user does. This will close the connection between the client and the server. The client sends a specific user identification number to the server.

Search route
In the beginning the user has to specify the departing station. But in the future the goal is that this will be done automatically with some kind of positioning. Then, the user has to specify the arrival station, if there are any preferred transportation types and if there are any disliked routes. So the input could be as follows: (From: Älvsjö, To: Kungsträdgården). The next thing that the user has to do is to send away the data to the server:

4.3.2. Content Provider

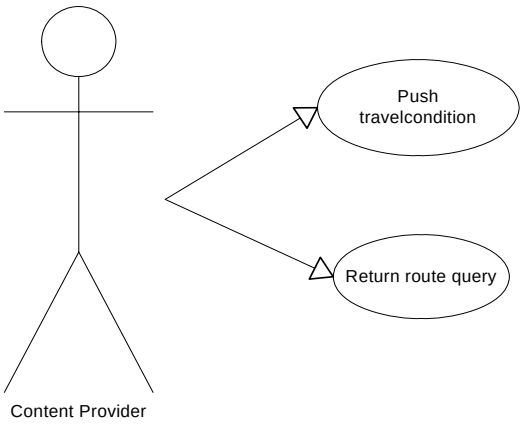

Figure 10: Use case: Content provider

The content providers, a part of whose is involved in the PIITSA project, supply the application with the traffic information. This information could be how a user gets from point A to point B, information about traffic jams or accidents, and information about the road conditions. The content providers will stream this information to the system, probably in an XML file or RSS stream. Therefore it doesn't matter who the content providers are, the application should work anyway, no matter how many content providers what are used or who they are. The content providers for this project are Swedish Road Administration [14], Stockholm Transport [13], Trafik Stockholm [11] and trafiken.nu [12].
The content providers does two things in our system, they push travel conditions and return route queries, as shown in figure 10.

Push travel condition
This is the key feature in the application, the feature that separates this product from other products. The content provider sends out information to the server. When something happens in the traffic, something that affects the travellers, then the server sends out a message to them. These things could be road construction, accidents or delays. The information in this message that will be sent is a short description of where it happened, which direction and how long it will last.

Return route query
This is just a simple answer to the query that our server sends to the content providers. The question will be in the format of “How do I get from point A to point B”. But there are some other things that the content providers do when this query is sent to them, i.e. check problems. What is meant with this is that when they select the best route, they see if there is some traffic problem in some part of the route. If there is, then that part will be excluded in the search. When this has been calculated and a matching route has been found, then this route will be sent to the server.

4.3.3. Service Provider

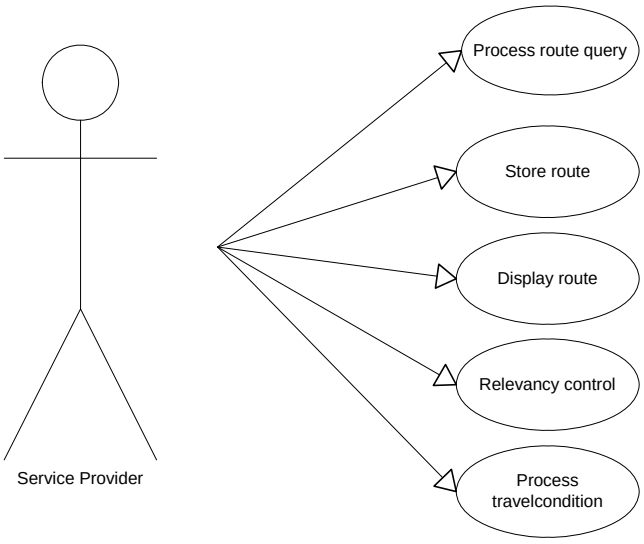

Figure 11: Use case: Service provider

This is the server core; it provides the service to the users, as illustrated in figure 11. This is the centre of the whole application. The server core forwards the data from the user and from the content providers. The server core is responsible to decide to whom the data is relevant. This is one of the most important features, without this there would be a lot of unnecessary data what would be transferred between the server and the users. With this feature only the users who needs this kind of information gets it. This will increase the application’s trustworthiness, as user profiles and problems are specified by the content providers and correlated with each other.

Process route query
This function forwards the route query from the user to the content providers.

Store routes
This is the function that stores the user's covered routes into the profile database on the server.

Display routes
This function retrieves the respond from the content providers, and forwards it to the specific user, which has made a query for a route. When the information arrives to the user it is presented on the PDA's display and the user can see the whole route.

Relevancy control
This is a very important function to minimize the amount of sent data between the user and the server. When there is some traffic information received at the server, this function performs a first filtering and checks the database on the server to see which user it concerns. Then it forwards the information to these users. This is a key function in this application.

4.4. Sequence Diagram

Figures in Appendix C.
All the sequence diagrams are based on the hybrid solution. It is, however, possible to use them with the centralized and decentralized solutions as well with some minor modifications.

4.4.1. Login Procedure

This will describe the dataflow for the login procedure and for the "load profile" procedure. The login process will be done automatically when the user starts the application.

- The TravelPlanner forwards the login information to Core. This is done by using NSBClient and NSBCore to get a connection.
- The Core checks in the database to see if the user has the right login information. If the users' username and password is found and matched against the database, then the user is registered in the database as active.
- The Core returns a message, containing specific user identification and the specific profile that belongs to the user, to TravelPlanner through NSBCore and NSBClient that the login procedure was successful.
- The TravelPlanner sends the received message to SortedLoginString, which performs necessary operations on the message.
- If the user has chosen to load profile, then will the message that is received from SortedLoginString be presented to the user.

4.4.2. Search Procedure

This will describe the dataflow for the search procedure. To start the dataflow the user must insert information in the search text field that is presented when choosing “Search new route” from the menu.

- The TravelPlanner forwards the search information to Core. This is done by using NSBClient and NSBCore to get a connection.
- The Core forwards the information to Search. Search processes the information and builds a query string which will be sent to the Content provider.
- The Content provider processes the query and returns a webpage as the result to Search.
- Search gathers the necessary data from the webpage and returns the information as a string to TravelPlanner through Core, NSBCore and NSBClient.
- TravelPlanner presents the result for the user and the user has the option to store the result in the user profile. The procedure for this will be explained below.

4.4.3. Insert Route Procedure

This will describe the dataflow for the insert route procedure. To start the dataflow, the user must insert information in the fields (name of the route, line, date and time), presented when choosing “Covered new route” from the menu or when choosing to add search result to the profile.

- The TravelPlanner forwards the information to Core. This is done using NSBClient and NSBCore to get a connection.
- The Core forwards the information to Stretch. Stretch processes the information and calls the correct method, i.e. InsertStretch. The method sends correct information to DBEncapsulation.
- The DBEncapsulation builds the query string and sends it to the database.
- The covered route will be inserted in the database. Core returns a message to TravelPlanner saying that it succeeds.
- TravelPlanner forwards the information to SortedLoginString. The information is the same that was forwarded to Core earlier. SortedLoginString process the information and adds it to the local profile.

4.4.4. Update Route Procedure

This will describe the dataflow for the update route procedure. To start the dataflow the user must update information in the text fields (name of the route, line, date and time), presented when choosing “Update covered route” from the menu.

- After the user has selected to update a covered route from the menu, TravelPlanner sends the local profile to MergeTypeString. MergeTypeString groups the profile into transportation types and returns the result to TravelPlanner.
- TravelPlanner presents the update field containing the information about the covered route that was selected. TravelPlanner sends the information that should be updated to UpdateString. UpdateString builds the string in the right format and returns it to TravelPlanner.
- TravelPlanner forwards the information to Core. This is done by using NSBClient and NSBCore to get a connection.
- The Core forwards the information to Stretch. Stretch processes the information and calls the correct method, i.e. UpdateStretch. The method sends correct information to DBEncapsulation.
- The DBEncapsulation builds the query string and sends it to the database.
- The covered route will be updated in the database. Core returns a message to TravelPlanner saying that it succeeds.
- TravelPlanner forwards the information to SortedLoginString. The information is the same that was forwarded to Core earlier. SortedLoginString processes the information and updates the local profile.

4.4.5. Delete Route Procedure

This will describe the dataflow for the delete route procedure. To start the dataflow the user must delete information about a stored route, presented when choosing “Update covered route” from the menu.

- After the user has selected to update a covered route from the menu, TravelPlanner sends the local profile to MergeTypeString. MergeTypeString groups the profile into transportation types and return the result to TravelPlanner.
- TravelPlanner presents the field containing the information about the covered route that the user chooses to delete. TravelPlanner sends the information that should be deleted to UpdateString. UpdateString builds the string in right format and returns it to TravelPlanner.

- TravelPlanner forwards the information to Core. This is done by using NSBClient and NSBCore to get a connection.
- The Core processes the information and forwards it to Stretch. Stretch processes the information and calls the method, DeleteStretch. The method sends information to DBEncapsulation.
- The DBEncapsulation built the query string and sends it to the database.
- The covered route will be deleted from the database. Core returns a message to TravelPlanner saying that it succeeds.
- TravelPlanner forwards the information to SortedLoginString. The information is the same that was forwarded to Core earlier. SortedLoginString processes the information and updates the local profile.

4.4.6. Event Procedure

This will describe the dataflow when an event occurs. The event can come from one of the content providers.

- Core checks for a new event at a given time interval by calling the class Event. The content provider returns the result of event, at least the latest.
- If there is more than one event received, Event processes each event and compares with the database through DBEncapsulation if it concerns any users. If so, the event will be sent to the user through NSBCore and NSBClient.
- TravelPlanner checks for received event at a given time interval, and if has received a new event it will be presented to the user.

4.4.7. Logout Procedure

This will describe the dataflow for the logout procedure. The logout process will be done automatically when the user exits the program by choosing “Exit program” in the menu.

- The TravelPlanner forwards the logout information containing the specific user identification, which was received when logging in, to Core. This is done by using NSBClient and NSBCore providing the connection.
- The Core checks in the database to see if the user has the right logout information. If the submitted information matches the information in the database, the user is set to no longer be active. If the information doesn’t match the database, nothing is changed there.
- The Core returns a message to TravelPlanner through NSBCore and NSBClient that the logout procedure was successful.
- The TravelPlanner receives the message and closes the application.

4.5. Database Design

Our application has two databases, one centralized placed on the server and one stored locally onto the PDA. The later is necessary since there are data and information that need to be stored on the PDA. It would be possible to store everything on the server, but that would lead to unnecessary amounts of transferred data. The whole database is not placed on the PDA for the same reason.

4.5.1. Database Server Core

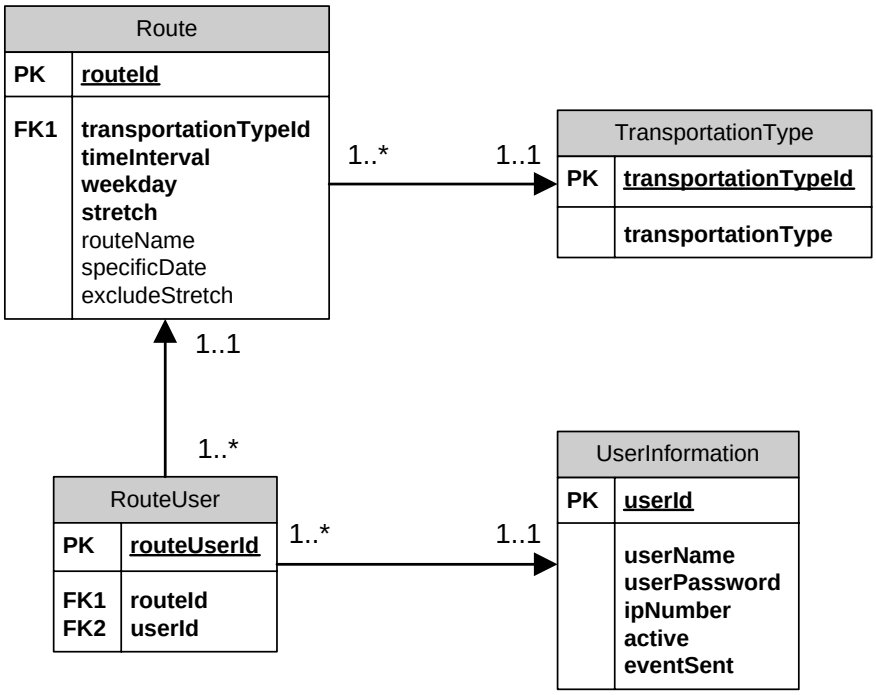

Figure 12: Database design: Server core database

Figure 12 shows the tables and their fields in the database. It also shows the relations, primary keys (PK) and the foreign keys (FK).

Route table
This table contains information about the routes that the users have searched for, or routes that the users want to cover. The information stored in this table is information about the route i.e. subway line, bus line etc. and not information about were the users start, stop or make changes on their routes. It has two relations, one to the RouteUser table and one to the TransportationType table. The relation to the RouteUser table is one-to-many (1..*), i.e. one post in the Route table can have one or several posts in the RouteUser table. This means that a user can have many routes, and since the route is unique because its routeId it could

only belong to one user. The relation to the TransportationType table is one-to-one (1..1), i.e. one post in the Route table must have one and only one post in the TransportationType table. routeId is the primary key in this table. The fields routeName, specificDate and excludeStretch are not mandatory. These fields can be used when the user wants to set a name on the route e.g. home, or wants information about a route just on a specific date or to specify if a route should be excluded.

TransportationType table
This table contain information about the different transportation types, i.e. subway, train, bus, car etc. It has one relation, it's to Route table. The relation is one-to-many (1..*). This means that one transportation type, e.g. subway, can belong to many routes, but however one route can only have one transportation type. The primary key is transportationTypeId.

RouteUser table
This table is only a help table between the Route and the UserInformation tables. It contains two fields with foreign keys to these tables. FK1 routeId is the foreign key to Route table and FK2 userId is the foreign key to User table. This table has relations to two tables, it has a one-to-one (1..1) relation to Route and a one-to-one (1..1) relation to UserInformation table. routeUserId is the primary key. The reason that there is a need for this table is that a user can cover many different types of routes, i.e. Subway could be covered on Mondays, and a specific bus line could be covered Wednesdays and Thursdays in the morning and evenings.

UserInformation table
This table has the information about the users stored. All the fields are mandatory; these are username, password, ipNumber, active and eventSent. This field is needed for the login process. ipNumber and active are to know where information should be sent, and eventSent is to know when that last event was sent in order to avoid duplicate events. The UserInformation table has relation to one table. It has a one-to-many (1..*) relation to the RouteUser table. This relation makes it possible for one user to have many routes, since RouteUser is a help table where the route and the user are combined. The primary key is userId.

4.5.2. PDA Database

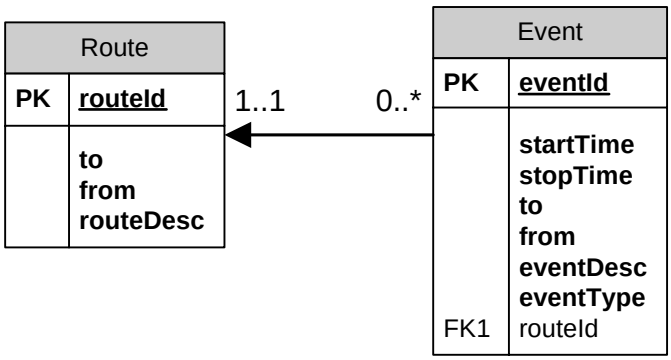

Figure 13: Database design: PDA database

Figure 13 shows the design on the database located on the PDA. The figure shows tables and their fields, relations, primary keys and foreign keys.

Route table
This table contains four fields. The to field will contain information about where the user wants to travel. The from field will contain information about where the user wants to start his route. In the last field, the routeDesc field, information about the route in general is stored. The Route table has a zero-to-many (0..*) relation to the Event table. This means that for one route there can occur zero or many events. In other words this means that an instance in the Route table could have one or more instances in Event table connected to it. routeId is the primary key in this table.

Event table
This table stores information about events that come from the content provider. These events could be crashes, delays or accidents. The information that will be stored is start and stop time of the event, which route the events belongs to, some description of the event and which type it is. There will also be a field which will contain a foreign key to Route table, this field is named routeId. The Event table has a one-to-one (1..1) relation with Route table, i.e. one event is only valid for one route. eventId is the primary key in this table.

4.6. Class diagram

4.6.1. Client

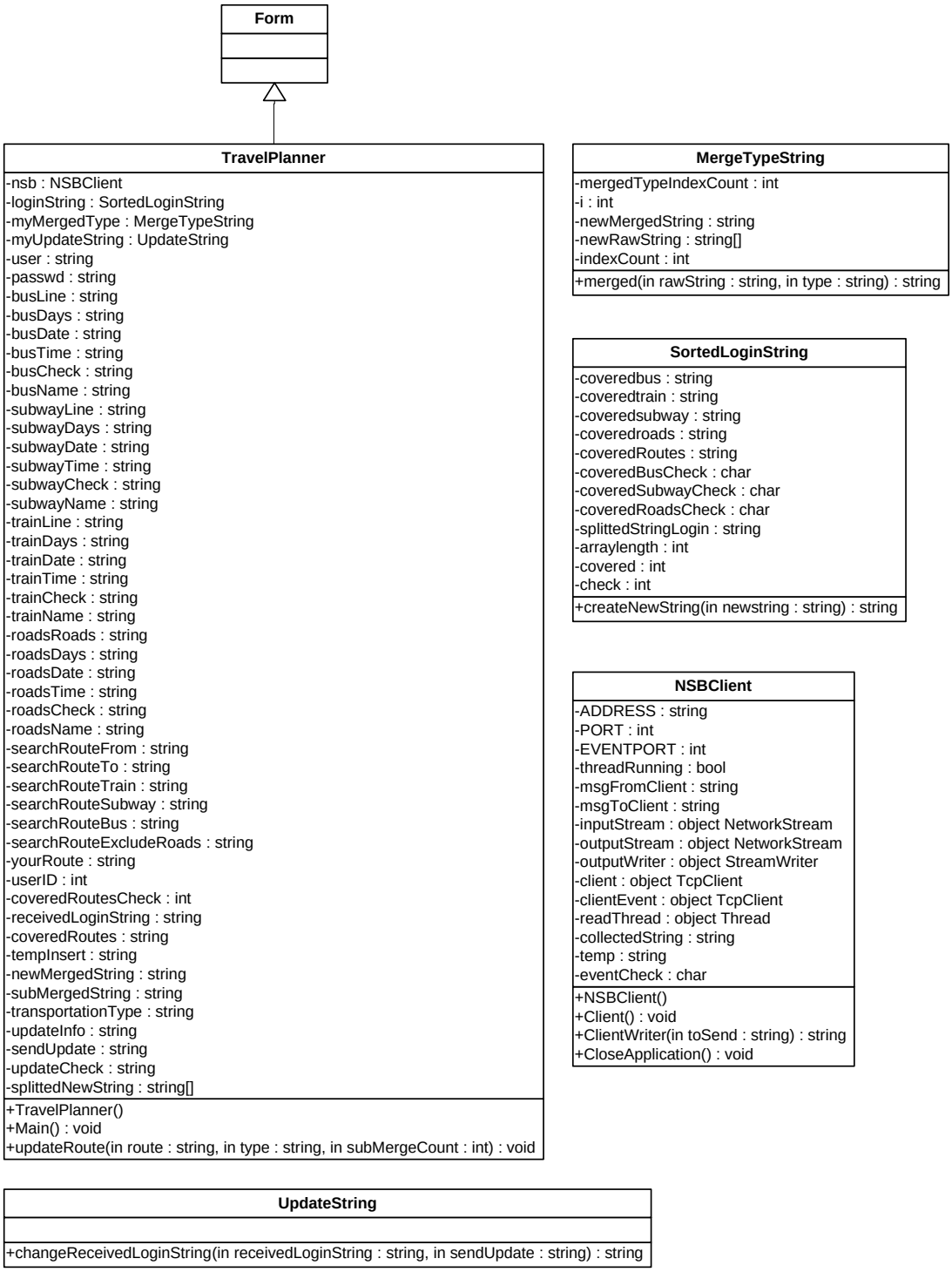

Figure 14: Class diagram: Client

Figure 14 shows the classes in the client side application. The diagram shows the classes attribute and the methods and it also shows the relations between classes, if there are any. On the client side there are six different classes. Two classes, Form and TravelPlanner, handle all the graphics on the PDA.

The Form-class is the main class there TravelPlanner-class is a part of the Form-class. The NSBClient handles all the connection against the server. Currently the connection is based on TCP/IP. This NSBClient should be replaced by a fully functional NSB (Network Selection Box) later on.

The class MergedTypeString is used when grouping the information on what kind of transportation type it belongs to. Class SortedLoginString is used to process the information received from the server when the user is logging in or when the user loads its profile. Class UpdateString handles updates made by the user, i.e. the user updates a covered subway line.

4.6.2. Server

On the server side there is also six classes as shown in Figure 15. The diagram shows the classes, attributes and the methods. Class NSBCore has the same task as NSBClient has on the client-side; it handles connections between the client and the server. The main difference between the two is that NSBCore listens for connections, while NSBClient take the initiative to connect.

Class DBEncapsulation handles all information regarding the database. It handles connections to, queries to and results from the database. The Core class is the main class on the server side. It processes the incoming data so that it is sent to the right place. It also contains a timer so that some task could be scheduled to be executed at a certain time. The Event class is called at a certain time from the Core class.

In the Event class everything about events or travel conditions is handled. It checks for new information regarding events and travel conditions, processes the data and makes it readable for the client. It also checks which user the event concerns. Class Search handles the search regarding how to get from point A to point B. It processes the incoming information received from the client, builds a query and sends it to one of the content providers. The received result from the content provider is processed to be readable for the client. The Stretch class handles information about the covered routes.

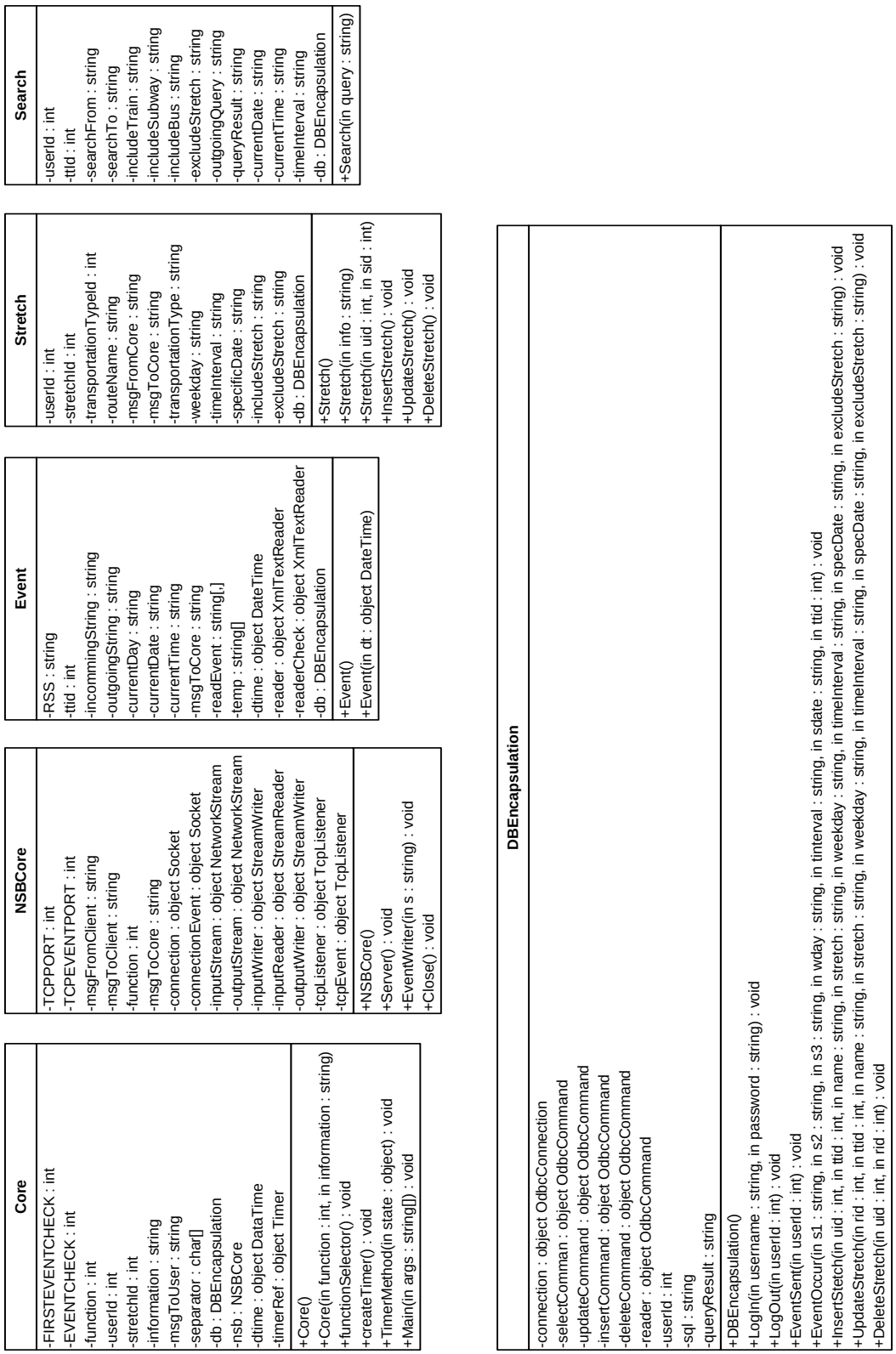
Core
-FIRSTEVENTCHECK : int
-EVENTCHECK : int
-function : int
-userId : int
-stretchId : int
-information : string
-msgToUser : string
-separator : char[]
-db : DBEncapsulation
-nsb : NSBCore
-dtime : object DataTime
-timerRef : object Timer
+Core()
+Core(in function : int, in information : string)
+functionSelector() : void
+createTimer() : void
+TimerMethod(in state : object) : void
+Main(in args : string[]) : void
NSBCore
-TCPPORT : int
-TCPEVENTPORT : int
-msgFromClient : string
-msgToClient : string
-function : int
-msgToCore : string
-connection : object Socket
-connectionEvent : object Socket
-inputStream : object NetworkStream
-outputStream : object NetworkStream
-inputWriter : object StreamWriter
-inputReader : object StreamReader
-outputWriter : object StreamWriter
-tcpListener : object TcpListener
-tcpEvent : object TcpListener
+NSBCore()
+Server() : void
+EventWriter(in s : string) : void
+Close() : void
Event
-RSS : string
-ttid : int
-incommingString : string
-outgoingString : string
-currentDay : string
-currentDate : string
-currentTime : string
-msgToCore : string
-readEvent : string[,]
-temp : string[]
-dtime : object DateTime
-reader : object XmlTextReader
-readerCheck : object XmlTextReader
-db : DBEncapsulation
+Event()
+Event(in dt : object DateTime)
Stretch
-userId : int
-stretchId : int
-transportationTypeId : int
-routeName : string
-msgFromCore : string
-msgToCore : string
-transportationType : string
-weekday : string
-timeInterval : string
-specificDate : string
-includeStretch : string
-excludeStretch : string
-db : DBEncapsulation
+Stretch()
+Stretch(in info : string)
+Stretch(in uid : int, in sid : int)
+InsertStretch() : void
+UpdateStretch() : void
+DeleteStretch() : void
Search
-userId : int
-ttId : int
-searchFrom : string
-searchTo : string
-includeTrain : string
-includeSubway : string
-includeBus : string
-excludeStretch : string
-outgoingQuery : string
-queryResult : string
-currentDate : string
-currentTime : string
-timeInterval : string
-db : DBEncapsulation
+Search(in query : string)
DBEncapsulation
-connection : object OdbcConnection
-selectComman : object OdbcCommand
-updateCommand : object OdbcCommand
-insertCommand : object OdbcCommand
-deleteCommand : object OdbcCommand
-reader : object OdbcCommand
-userId : int
-sql : string
-queryResult : string
+DBEncapsulation()
+LogIn(in username : string, in password : string) : void
+LogOut(in userId : int) : void
+EventSent(in userId : int) : void
+EventOccur(in s1 : string, in s2 : string, in s3 : string, in wday : string, in tinterval : string, in sdate : string, in ttid : int) : void
+InsertStetch(in uid : int, in ttid : int, in name : string, in stretch : string, in weekday : string, in timeInterval : string, in specDate : string, in excludeStretch : string) : void
+UpdateStretch(in rid : int, in ttid : int, in name : string, in stretch : string, in weekday : string, in timeInterval : string, in specDate : string, in excludeStretch : string) : void
+DeleteStretch(in uid : int, in rid : int) : void

Figure 15: Class diagram: Server

5. Results

In this chapter the results will be presented. Some measurements on the data traffic were made. Also, the cost for the application will be discussed in this chapter. Another item regards the information sources. Finally the system design and the services of the application are presented.

5.1. Measurement of Data Traffic

Some rough measurements of the traffic volume from/to the PDA were carried out. Data transfers are triggered by window shifts. There, we can compute the amounts of sent and received data before before and after completition of a certain action. As the goal is to get some qualitative overview of measurements rather than to get exact quantitative results, the measurement was carried out three times per action and the result were averaged. The results were then rounded off.

The reason for that only measuring the data on the PDA, is that this amount of data is the most important for the user. This is the data that the user currently has to pay for. So, of course, one of the goals was to minimize this amount of data.

Figures from the Measurement
Note that these numbers are of rather qualitative nature.

Starting the application
Data sent: ~400 Bytes.
Data received ~600 Bytes.
Most data that are sent here relate to the connection establishment between the PDA and the server core.

Login procedure
Data sent: ~300 Bytes.
Data received: ~650 Bytes.
When the user clicks on connect to service in the start menu, his information is automatically sent to the server. The data that are sent are a number (to tell the server that a login procedure has started), the user's username and password. Then the PDA will receive data about the user's stored covered routes. This is made like this to make it smoother to use the application by minimizing the waiting time for the user.

Event arrived
Data sent: ~0 Bytes.
Data received: ~350 Bytes/event.

There is no data sent from the user when the user receives an event. But the user receives roughly 350 Bytes per event. Most of the time there arrives more than one event at the time; therefore the user will receives a larger amount of data when this happens.

Load profile
Data sent: ~100 Bytes
Data received: ~650 Bytes.
The data that are sent when the user chooses to load profile from the menu is almost the same as when the user logs in. But load profile procedure doesn't send as complex information as login does. The respond is more alike, both procedures receive the user's covered routes from the server core database.

Search Route
Data sent: ~400 Bytes.
Data received: ~450 Bytes.
Information about the search is sent to the server core database. This information addresses the type of transportation, the line/road, and day and time.

Insert covered route
Data sent: ~400 Bytes.
Data received: ~200 Bytes.
Almost the same information as in search route is sent here, which implies the small amounts of sent and received data.

Update covered route
Data sent: ~400 Bytes.
Data received: ~200 Bytes.
The information as in search route and insert route are sent.

5.2. The Cost of the Service

In the solution that was created, there are almost comparably low costs for the user. The information data from the content providers are free, and according to the result presented above there are not much data transferred between the server and the PDA.

In the thesis work, this application will be free, but if it should be commercial, cost for the user could arise when the user purchases the application. It shouldn't cost anything for the user to use the application, except for the data traffic. If it will cost money to use the application, then the amount of users might not be quite as high as if it was for free.

5.3. The Information Sources

The traffic information that is used in our application is collected from trafiken.nu [12] RSS (Rich Site Summary) site. Here, information about for example accidents, delays etc is received.

The information about public transportation is received from the Stockholm Transport web page. Here the server application receives information about the routes that the users search for, as well as information about departure and arrivals.

The application has no positioning system, which was considered to be beyond the scope of this work. It would have taken too long to learn the necessary skills to use any of the standard technologies. Moreover, it would have incurred extra cost.

In an early stage of this thesis work there were some plans to perhaps use DAB in our application. But it was decided to not use this technology. One major reason for this decision was that the Swedish government has decided to cancel the deployment of DAB. However, as new standards might emerge and be deployed, they could be incorporated, which will be handled by the NSB.

5.4. Chosen System Design

In the planning phase three different system designs were developed, a centralized one, a decentralized one and a hybrid solution. The design that finally was chosen was the hybrid solution. The main reasons for this decision were:

- It is the smartest solution; there is intelligence both on the server and on the PDA.
- It is the most secure solution for the future. If the application will be commercial, it's quite easy to develop it further.
- It makes it possible for the NSB to choose the best available connection for the moment.
- There won't be much data traffic between the PDA and the server, only relevant data will be transferred.

So the solution was to create a free-standing application that can run in the background when the user is listening for new events. If the user wants to create a new covered route or search for a new route, just bring the application to the front.

When the implementation of the application started, the design of the database had to be redesigned. Currently no database was needed on the PDA, but there might be one needed in the future.

5.5. Services of the Application

The application resulted in three major services. These are cover route, search route and receive event. The most powerful service of these is the receive event with up-to-date traffic information.

Cover Route
This will probably be the service that is mostly used. Here, the users can specify which part of a route they want to cover (i.e. receive traffic information about). They just have to input some data about the route. These data are a name for the route, which type of transportation, which line, which day/date and which time the cover are for.

There is one other way that a route can be inserted here. It is when the user searches for a route from point A to point B. Then will the different part of this route be included here. If the route has more than one part, for example if the route includes both a bus ride and traveling with the subway, then these two parts will be saved separately. This is done to protect the user's integrity, now only the user will know his whole route.

Search Route
This is an important service that TravelPlanner provides. But it isn't a unique service. There are several services out there that are almost the same. This is a straight forward service: The user types in the name in the departure station and the destination. This query is forwarded to SL's homepage [13], which calculate the best route for the user. The information about the route is then displayed on the PDA; there the user can see it. Then the user has two options, either accepting the given route (which will be included in covered routes) or search for a new route.

Receive Event with Traffic Information
This service makes TravelPlanner unique; it is the "added value" of the application. Without this service would TravelPlanner be just another ordinary travel guide application.

The users will receive information if something happens on their covered routes. The information that they receive are information such as; which route it concerns, which direction, which date/day and how long it will go on. This information is gathered from the trafiken.nu [12] homepage and forwarded only to those users that it concerns. These users are those who have this specific route in their covered routes.

When the users receive this information, they can chose to search for a new route that will exclude the “damaged” route, simply by clicking on a button in the application. The rest is handled automatically and TravelPlanner contacts SL’s homepage [13] for a new route. This will be automatically forwarded to the user and be displayed on his PDA. At the moment there is no part that handles the exclusion of “damaged” routes. But in the future this will be done by SL [13].

The application is designed so that the user will never miss any events when being logged on to the service. If the user doesn’t read the events, they will simply be stored in a queue so that the user can read them later. The application is designed to check for new events every other minute, but it is configurable so it could be set to any interval.

6. Problems

As in all application development, some problems were experienced. The most important of these problems that were found are listed below, and short explanations of each problem and its solution are also listed.

6.1. Compact Framework

Since the application was developed for a PDA, the standard Microsoft .Net Framework couldn't be used. It is made for computers and servers, but not compatible with PDA's. The application had to be developed in an environment which is called Smart Device Application in Microsoft Visual Studio .Net 2003. But to do this, Compact Framework had to be used.

Microsoft scaled off several features when they created Compact Framework. This caused several problems, but the most irritating one was that almost all information and help available on the Internet was created for the standard .Net Framework.

After a while, another problem was discovered. The emulator in Visual Studio on the PC didn't want to run the application anymore. It complained on a part in the code that didn't contain any errors. So the solution was to stop using the emulator and simply build the application and debugging it when an error has occurred. This solution made it possible to run the application on the computer instead on the PDA, which was quite time consuming. So this was a big plus.

6.2. Exiting the Application

This problem wasn't encountered until in the end of the programming phase. Compact Framework doesn't support the same exit commands as the standard Framework does, especially when working with threads.

When working in standard Framework there are methods like Enviroment.Exit(0) that will close the whole application and everything that concerns the application. Standard Framework handles threads much better, for example there is a method called join, which blocks the thread so that it is possible to kill it.

These methods aren't available in Compact Framework since the application has to be able to run in the background. There is an add-on available called OpenNETCF which handles threads much better than Compact Framework. But this was found out too late, there was not enough time to learn the necessary programming skill to manage to control this add-on. Its use should be a part of future work.

Finally a solution to this problem was found. It is possible to force the threads to close with the help of a while statement.

6.3. Positioning

The positioning issue was discovered quite early. Either it would cost too much to find a solution or it would result in too much work.

With the equipment that was available, the only method to solve this problem was to use MPS, but it was too complex to learn in this thesis work.

So the solution that was chosen was instead to force the user to input his starting position in the same way as the user does for the arrival station.

6.4. Database Connectivity

MySQL was chosen as the database in this application. But since this was chosen, some problems with the connection were discovered. No predefined ODBC (Open DataBase Connectivity) in Visual Studio was found. There were some problems to implement a specific ODBC that was made for just MySQL, the one that was finally chosen too use was the MySQL ODBC 3.51 driver.

This problem was not too hard to solve, but still it stalled the development a little.

6.5. Mobile Database

In the planning phase a database was decided to be used on the PDA and not only on the server. But after some research on the Internet [15], some information was discovered that there were not many good, free solutions available.

It was decided to use arrays instead of a local database. This solution resulted in some more programming work, but it was worth it.

6.6. PDA Problem

The PDA, a Compaq iPAQ H3900 with Pocket PC installed, caused some problems. The reason for this is that it wasn't a very new model.

The first problem that was noticed was the connectivity. The PDA has a jacket which should support the PDA with a GPRS connection. It should but it didn't. This was tried several times but it never worked. The solutions for this was quite simple, just change to another jacket.

Another problem with the PDA was that applications hung in different way when they were run. This made it really hard to try to find any solution to this problem. If an application hangs when doing one thing, it wasn't sure that it would hang the next time, not even if doing the exact same thing. This problem was really annoying, there was no way to know if the code was wrong or it was just the PDA that had a bad day. The only thing to do when this occurred was to either restart the PDA, or sometimes it was enough to just kill the process that belonged to the application.

6.7. Search Query to Stockholm Transport

This was also a problem that was discovered when there wasn't much time left to correct. When a search queries to SL's homepage was forwarded, it was necessary to spell the name of the station in an exact way. Because one part of this problem is that a third party is involved. SL's service can't be changed, nor can SL be asked to change it

For example, if the user wants to travel from Kungsträdgården to Älvsjö, the user had to type in Kungsträdgården Hamngatan and Älvsjö Station. If the user doesn't do this, then the user won't get any hits from SL's homepage.

If there were enough time, then this would have been fixed so that it would be better. It is possible to use the result and send a new query, to SL, based on the first result.

The way the application makes its search query for a route forced us to learn how to strip an html site. Since the search query is forwarded to SL's homepage there was not any way to directly use that information. Something has to be done with the information, so that it is possible to use it in the application.

The solution looks like this:
When the user makes the search, a C# class called Webclient starts to collect the data from SL's homepage. The relevant text was extracted from the HTML page

based on its tags. An HTML site contains even more information that is unnecessary for the application than an XML site does.

6.8. Problem with the Event Reception

This problem appeared sometimes while testing the application. It didn't happen every time, or only on one machine so any reason of this problem wasn't found. When the application was running and the user has logged into the system, then the application displayed an error message instead of the event as it was supposed to.

The reason of this problem was that the application tried to work with an empty string. This was easily solved by creating a check method to see if the string was empty before its processing started.

6.9. Collecting Traffic Events

The traffic events were collected from trafiken.nu. They are providing a RSS file on their website. This RSS file is programmed in XML. The application can't just use this code in the original form, and then it would end up with a bunch of worthless text i.e. tags etc. So the useful information has to be extracted from the code. Another reason for this is to minimize the traffic load.

7. Discussion

Developing a mobile application is very interesting. The design part of the application was very time-consuming and was changed several times. The goal was to make it as user-friendly as possible, and it also had to fit the requirements from both Saab AB and from Blekinge Institute of Technology. From the design document we implemented the application in C# using Microsoft Visual Studio. While implementing we've noticed some difficulties. One of them was developing against a PDA. We had to use Compact Framework instead of the usual framework, which is used on regularly computers, to solve the problem. There was a few problems regarding involved third parties, for example the search query to SL and traffic message from trafiken.nu. Solving these problems was more time-consuming than solving other problems. To get a graphical user interface to work smoothly we had to try to think like a user. If a user won't feel being in control of the application, then it won't be used. So getting menus and information to contain right text and at the same time be user-friendly requires a lot of thinking and creativity.

The application are finished resulting in a working product which solves most of the tasks that we defined in the introduction. The application has been tested by the developers, and also other persons have been asked to test the application as well in order not to overlook any problems. This was made to be sure that the application was working correctly. The reason that we brought in other persons was, that it is easy to be blind if you are testing your own application. Since the developer knows how to work with the application, many shortcoming errors will probably be missed.

These testers were testing the application on the PDA, but some also tested it on the computer (which was easier for them, as not all of them have previous PDA experience). They were creating covered routes, made some searches for routes and received events with traffic information. In other words, they were using all the functions that the application provides.

We also had Maria Jernbäcker at Saab AB viewing the application to get some response from a non-technical person but with considerable experience with this kind of applications. This resulted in some smaller changes in the structure of the menus and renaming some options.

When all these tests were made, we had received some responses being the background for this discussion. Most of the response was positive, but we have found some areas that could be improved if the application would be commercialized.

Some of the future works that can be done are:

Develop a better search function that isn't depending on SL's database.
Even if this isn't fixed, the application will work. This future work will relieve the users from typing the exact name of the stations.

Make the application more suitable for multiple simultaneous users.
This is the most important problem to fix if the application should become commercial. Currently, only one user can use the application at the same time. But since it is only a demonstrator, this isn't vital for the application to work.

Include more traffic information like parking lots, flight and train traffic, and make the application cover whole Sweden.
This can be done to get a larger target group for the application. If this information will be included, then more users are likely to explore this application and its services.

8. Conclusion

In this thesis work an application for mobile units, i.e. PDAs, was created. The application is a part of the VINNOVA-sponsored project PIITSA. The application should be used in a demonstration purpose, where the functionality of the network selection box should be shown. To create the application, several steps in the development process have to be done, from pre-study to implementation and testing. The application is a smart travel planner. It should be able to give answers on “How I get from point A to point B in the best way” but it should also be able to serve its users with information about disturbances in the traffic, i.e. on roads, public transport etc that concerns their routes.

After the pre-study was made, a system design was developed. Also the different kind of services and integration against different content providers was done in the implementation phase. The finished application has three main services:
(1) search route from point A to point B, when traveling with public transport,
(2) profile handling so the user can choose what routes they want to cover, and
(3) receive information about if something happens on a particular route. This means that the system will notify the concerned users about traffic conditions and delays etc.

9. Future Work

The application is now finished, at least enough for a demonstration. But there is more work that could be done. It could be to come up with more services, for example let the user find available parking lots. It could also be making national covering of the services, enable positioning etc.

Search Function
This is one thing that has to be rebuilt. As it is now, the user has to type in the exact name on the station. If the user doesn't, then the application won't find any station in the SL's database. Another way is to use another database.

Multiple simultaneous users
This is a major problem to fix if the application should be published. As it is now, the server can't handle more than one user at the time. This is of course not feasible if this application should be sold to people.

Available parking lots
This service could be built into the application. Then the user would see if there is any meaning for him to take the car to city. This service will especially be appreciated by the "multimodal" travelers who use several types of transportation to go from point A to point B. With this service they could take the car to the city border, to empty parking lots, and continue the travel with the subway or something else.

National coverage
The application could be expanded so that it would cover most of the country, or even go international.

Flight
Flight could also be included in this application. This could make the travelers even more "multimodal".

Long-distance Train
Long-distance trains could easily be included in this application. Banverket has information about departures, arrival and delays etc.

Notification on Received Event
A notification like a beep or some other sound can be implemented so the user won't miss an event as soon as it is received.

NSB integration
The inclusion of the NSB will make the application independent of which kind of network that is available.

10. References

10.1. Paper References

[1] Larman, C. (1998) Applying UML and patterns, an introduction to object-oriented analysis and design. New Jersey: Prentice Hall

[9] Lundhem, S. (2002) Att lyckas med ett webbprojekt. Bilda förlag

10.2. Digital References

[2] GSM World – What is GPRS. GSM World. Available on Internet http://www.gsmworld.com/technology/gprs/intro.shtml#1 [Collected 2005-12-07].

[3] MySQL AB. Available on Internet http://www.mysql.org [Collected 2005-12-07].

[4] Visual C# Developer Center. Microsoft. Available on Internet http://msdn.microsoft.com/vcsharp [Collected 2005-12-07].

[5] A simple .Net TCP Socket Component. The Code Project. Available on Internet. http://www.codeproject.com/csharp/XYNetSocket.asp [Collected 2005-12-07].

[6] What is PDA? Webopedia Computer Dictionary. Available on Internet. http://www.webopedia.com/TERM/P/PDA.html [Collected 2005-12-07].

[7] MSDN Library. Microsoft. Available on Internet. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnce30/html/ppcfaq.asp [Collected 2005-12-07].

[8] Visual Studio Launch. Microsoft. Available on Internet. http://msdn.microsoft.com/vstudio/previous/2003/faq/default.aspx [Collected 2005-12-07].

[10] ConnectionStrings. Available on Internet. http://www.connectionstrings.com [Collected 2005-12-13].

[11] Trafik Stockholm, Available on Internet, http://www.trafikstockholm.com [Collected 2006-02-21]

[12] trafiken.nu, Available on Internet, http://trafiken.nu [Collected 2006-02-21]

[13] Stockholm Transport, Available on Internet, http://www.sl.se [Collected 2006-02-21]

[14] Swedish Road Administration, Available on Internet, http://www.vv.se [Collected 2006-02-21]

[15] SQLite, Available on Internet, http://www.sqlite.org/ [Collected 2005-12-07]

11. Index

A

Abstract 3
AerotechTelub 3
Analyze 11
Application 7, 11, 18, 56
Automatic 24

B

Billing 13
BTH 3, 7
Busses 19

C

C# 56, 68
Centralized 26
Class 45
Class diagram 44
Client 44
Combitech 2, 56
Community trains 19
Compact Framework 52, 56
Competitive 14, 20
Connect 15, 34
Connectivity 53
Content Provider 12, 35
Contests 4
Cooperation 7
Cost 12, 20, 48
Cover Route 16, 34, 50

D

DAB 30, 49
Database 16, 29, 41, 43, 53
Dataflow 28, 37
Decentralized 28
Defined 12
Delay 7
Delete 39
Demand 17
Demonstrate 7
Design 10, 18, 26, 33, 41
develop 12, 13, 14, 56
Development 12
Disconnect 34

E

Education 11
Element 18
Encrypted 20
Eniro 21
Event 27, 31, 40, 47, 55

F

Feasibility study 10
Feature 59
Fee 13
Fiedler 2, 7
Figures 6, 47
Flight 59
Future 57

G

Galileo 33

Gender 11
Goal 8, 33
GPRS 7, 27, 54, 69
GPS 33
Group 16

H
Hybrid 30, 33

I
Implementation 10, 11
Important 11
information 12, 49, 55
Information Sources 19
Insert 38
Intelligent 31
Introduction 56
ITS 2

J
Jernbäcker 2, 7, 56

L
login 37
Logout 40

M
Measurement 47
Menu 18
Microsoft 52, 56, 69
Mobidock 23
Mobile 53, 56
Mobile Services 8
MPS 33
MySQL 69

N
National cover 59
Needs 10

O
Offers 16
OpenNETCF 52

P
Parking lots 59
Parties 7
Payment 12, 13
PDA 3, 7, 16, 25, 28, 43, 49, 52, 54, 56, 69
PIITSA 3, 7
Planning 26, 29, 30
Pocket PC 69
Positioning 24, 53
Powerful 16
Pre study 10
Prerequisites 10, 15
Procedure 12, 37
Profile 16, 37
Public Transport 19
Pushed 24

Q
Query 36

R
Receive Event 16, 50
Roads 19

Route 16, 36, 39, 41
RSS 49, 55

S
Scalability 31
Seamless 7
Search 35, 38, 54, 59
Search Route 15, 35, 50
Security 13, 20
Sequence Diagram 37
Server 45
Server Core 41
Service 10, 13, 15, 16, 48, 50
Service Provider 36
Simultaneous users 59
SL 7, 19, 22, 25
SMHI 12
SMS 7
Solution 17
Stockholm Transport 54
Stream 19
structure 10, 11, 17, 32
Subgroups 16
Subways 19
System 26, 49

T
Target 16
Target group 11
TCP 71
Technical 56
Technology 17
Technology skill 11
Thanks 2
Thesis 8
Third party 12
Traffic 47, 55
Traffic Information 19, 20
trafiken.nu 19, 20, 25, 49, 55
Transportation 34
Travel 15, 19
TravelPlanner 15, 20, 50

U
UMTS 7, 27
Unique 10
Update 39
Use Cases 33
User friendly 10

V
Vägverket 7
Västtrafik 23
VINNOVA 3
Visual Studio .Net 56, 69
VV 19, 25

W
WAP 7
WLAN 7

X
XML 55

Appendix A – Glossary

DAB	Digital Audio Broadcast
DB	Database
GPRS	General Packet Radio Standard
GPS	Guided Position System
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
KBPS	Kilobit Per Second
MMS	Multimedia Message Service
MPS	Mobile Position System
NSB	Network Selection Box
ODBC	Open DataBase Connectivity
OEM	Original Equipment Manufacturers
OS	Operative System
PC	Personal Computer
PDA	Personal Digital Assistant
PIITSA	Personal Information Intelligent Transport systems through Seamless communications and Autonomous decisions
PUL	Person Uppgifts Lagen
RSS	Rich Site Summary
SL	Stockholm Transport
SMHI	Swedish Meteorological and Hydrological Institute
SMS	Short Message Service
VB	Visual Basic
VV	Swedish Road Administration
WAP	Wireless Area Protocol
WWW	World Wide Web
XML	eXtensible Markup Language

Appendix B – Explanation of Terms

Application
The program that will be run on the PDA.

C#
C# is a part of the concept Microsoft .Net. C# is a programming language that has many similarities with other languages like Visual Basic, C, C++ and Java [4]. The reason that C# was chosen for development of the application, was that it is fairly easy to create graphical interfaces and the PDA that was used in the project supports applications programmed in C#.

Class Diagram
In a class diagram, the developer can sketch the program before starting to create it. A class is a part of the program that is used in object oriented programming. A class diagram helps the programmer keep control of the data structures and methods. In a class diagram are all classes drawn and the relationship between them is shown.

Content Provider
Are those providing the content that we are using. With content we mean the information about traffic and the information of how to get from point A to point B.

DAB
Digital Audio Broadcasting, also known as digital radio.

Database
A database is used to store data in an organized way, so that all data are easy to retrieve and use/update. There are many different kinds of databases available today. The one that we are using in this thesis is called MySQL.

Event
With event we mean the information we get from trafiken.nu when something happens in one of the transportation systems. It is this information that is forwarded to the concerned users. This creates the most powerful service that the application provides.

Feasibility Study
Feasibility study is a study that is made before the creation of the program. It is made to make it easier for the creator when starting to build the program. In this pre-study, things like background, existing services, and demands and so on are looked more closely into.

GPRS
GPRS stands for General Packet Radio Service and is a non-voice service that allows information to be sent and received across a mobile telephone network. The theoretical speed of GPRS is 171.2 kbps using all eight timeslots at the same time [2] but the reality is quite far from this. The reason for using this technology was mainly that the PDA has a jacket with GPRS and GSM available on it and the fact that the Network Selection Box wasn't developed. But also the cost and speed was important. It would cost too much to use SMS, and GSM would probably be a little to slow and may even be to expensive to establish a data connection.

Microsoft Visual Studio .Net 2003
"Microsoft Visual Studio .Net 2003 is the comprehensive, multi-language development tool for rapidly building and integrating XML Web services and applications. Visual Studio .NET 2003 offers a highly productive environment, in which to develop a broad range of Microsoft .NET-connected applications and technologies. Using the high-performance Microsoft .NET Framework run-time environment, Visual Studio .NET providing powerful tools for designing, building, testing, and deploying Web services and applications, as well as sharing best practices and guidelines in a team environment." [8]. This is the developer tool that we used to create our application.

MySQL
MySQL stands for My Structure Query Language and is one of the world's most popular "open source" databases. The reason why we chose MySQL is that it is a free program. It is quite easy to use, and both of us had some experience with this database. It is platform independent; it can be run on over 20 different platforms [3].

PDA
"Short for personal digital assistant, a handheld device that combines computing, telephone/fax, Internet and networking features. A typical PDA can function as a cellular phone, fax sender, Web browser and personal organizer. Unlike portable computers, most PDA's began as pen-based, using a stylus rather than a keyboard for input. This means that they also incorporated handwriting recognition features." [6]. This is the client in the system, enabling the user to use the service. The program should work with any kind of PDA that has Pocket PC as its operating system, as long as it could connect to the Internet in some way.

Pocket PC
Pocket PC "is software for PDA's. Pocket PC has both OS and application components bundled specifically for PDA's. These include a set of system components from the Windows CE OS and application components.

Windows CE is a set of components that an OS developer uses to build a proprietary OS. OEM's who purchase Windows CE use Platform Builder, an

integrated development environment, to assemble and compile an OS image. The image includes Windows CE and custom components, and custom device drivers. The OS image is then programmed into the read-only memory of a microprocessor chipset. Windows CE is not restricted to Pocket PC devices. It is appropriate for cash registers, cellular telephones, medical instruments, and any other appliance that requires a small, embedded OS.

The Pocket PC OS is a custom version of Windows CE developed by Microsoft. It contains APIs and applications that are not included in Windows CE 3.0 itself. Likewise, Windows CE contains components that are not included in the Pocket PC OS." [7].

Public Transport
Public transport means the different kinds of transportation that are available, such as bus, train, subway, etc. Individual traveling by cars, motorbikes etc. is not included.

Prerequisites
A prerequisite means something that is demanded to make another thing possible.

Route
A route is the whole way from point A to point B that the user will travel. A route is created when the user want to get information on how to get to a certain place. A route consists of one or several links or stretches

Service
A service is what the application is offering the user. In this application, the user can search routes, create profiles and receive related events.

Service Provider
A service provider is a third party that provides a service. Trafiken.nu is a service provider to us, but in the same time TravelPlanner is a service provider to the users, through the application.

Sequence Diagram
Is a diagram that describes the data flow through the application. This diagram is made before the programmer starts to create the application. This is a part of his "roadmap" that will be used to create the application.

Stream
When some data are sent out to one or more users, this technique can be called streaming data.

Target Group
A target group is a group of people that are similar in some way. For example, they are between 30 and 40, men and have a PDA. That man who fits in this description belongs to this target group.

TCP-Socket
"A socket is an end point in network communication; it is uniquely identified by an IP address and a port number. TCP/IP is the most common protocol used for inter-process communication. In many cases it is also the most efficient way to send data between client and server programs. The parties communicating via TCP socket can live on different machines on the network, but they can also live on the same machine or even within the same process." [5]. This is used to send data between the server and the client using GPRS.

TravelPlanner
TravelPlanner is the name of the application that was created in this project work.

Use Case
A use case is created in the beginning. It is made to make it possible for the creator to find out what the users will be willing to do. A use case shows which functionality the user has in different phases in the application. It is also a part of the creators "roadmap".

Appendix C – Sequence Diagrams

Table of Figures

Figure 16: Sequence Diagram: Login **74**
Figure 17: Sequence Diagram: Search Route **75**
Figure 18: Sequence Diagram: Insert Route **76**
Figure 19: Sequence Diagram: Update Route **77**
Figure 20: Sequence Diagram: Delete Route **78**
Figure 21: Sequence Diagram: Event Arrived **79**
Figure 22: Sequence Diagram: Logout **80**

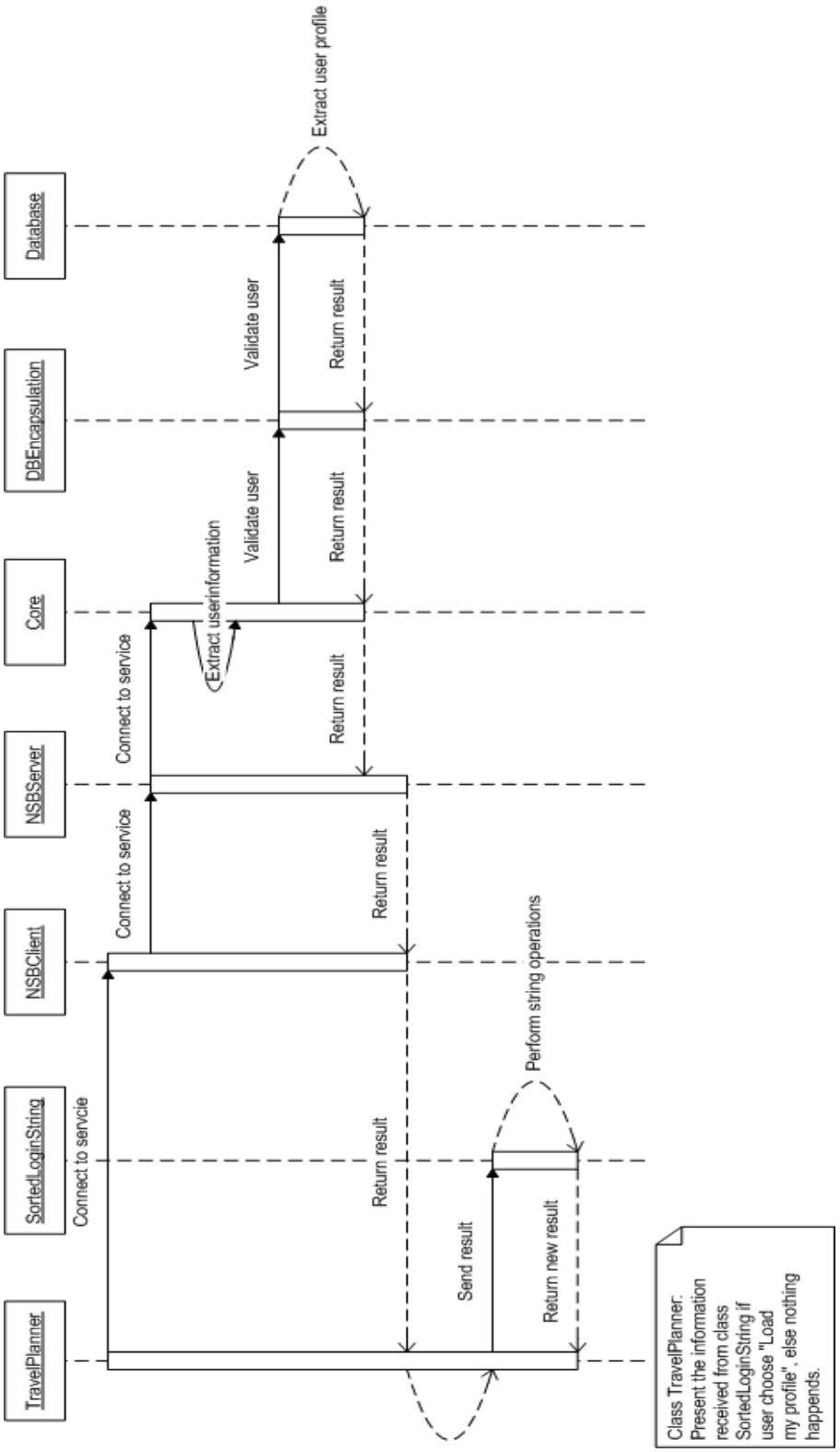

Figure 16: Sequence Diagram: Login

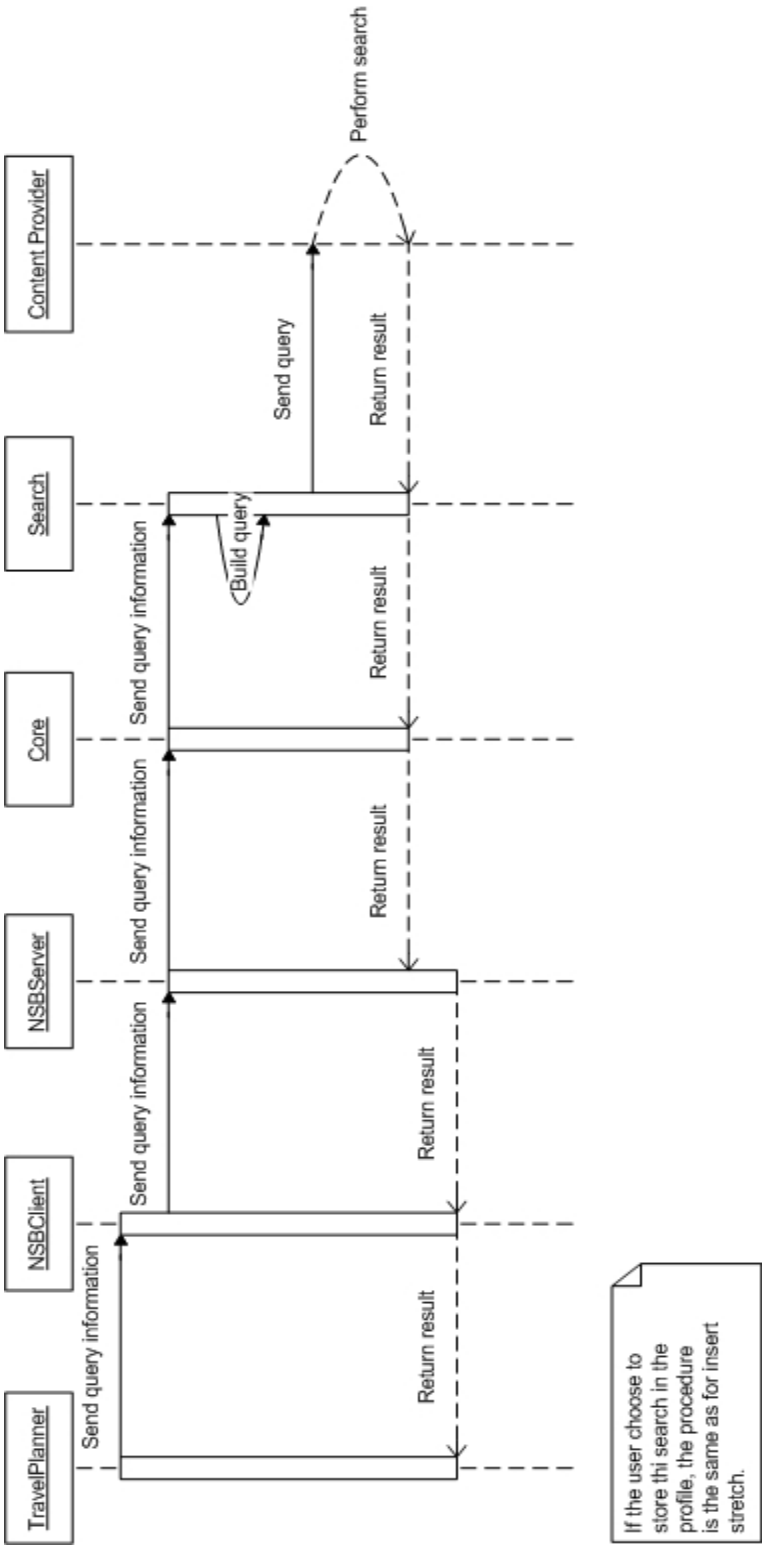

Figure 17: Sequence Diagram: Search Route

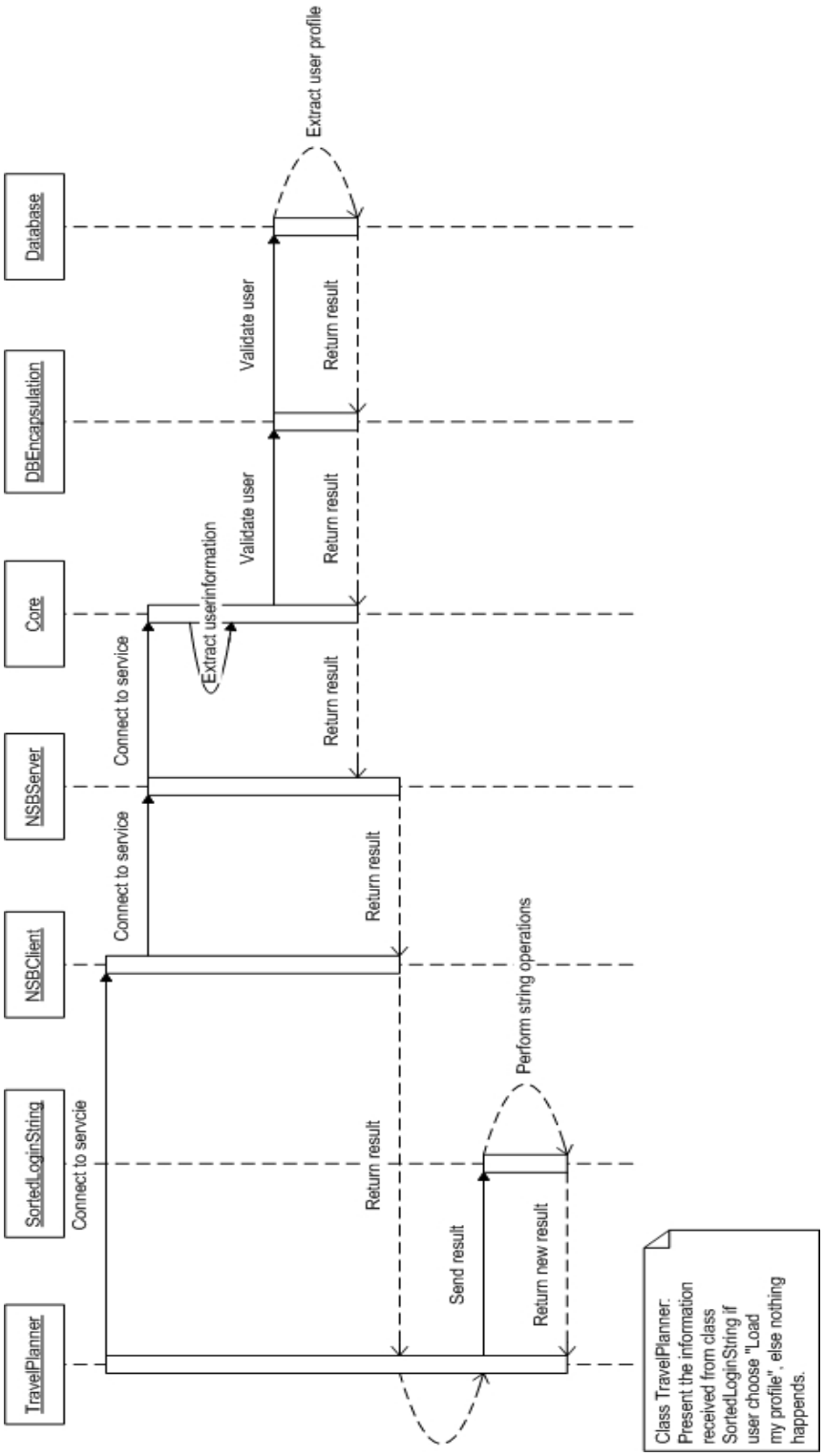

Figure 18: Sequence Diagram: Insert Route

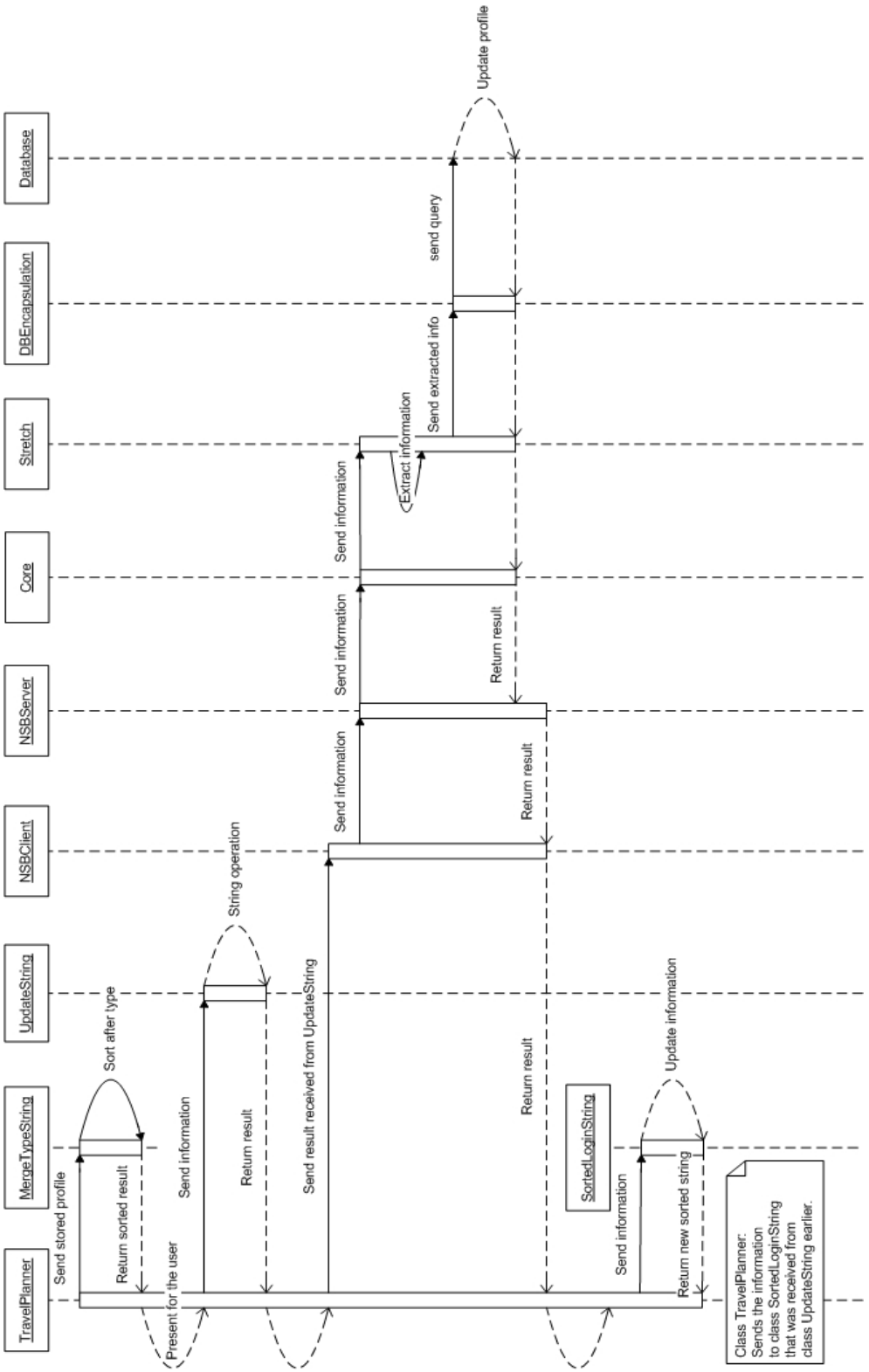

Figure 19: Sequence Diagram: Update Route

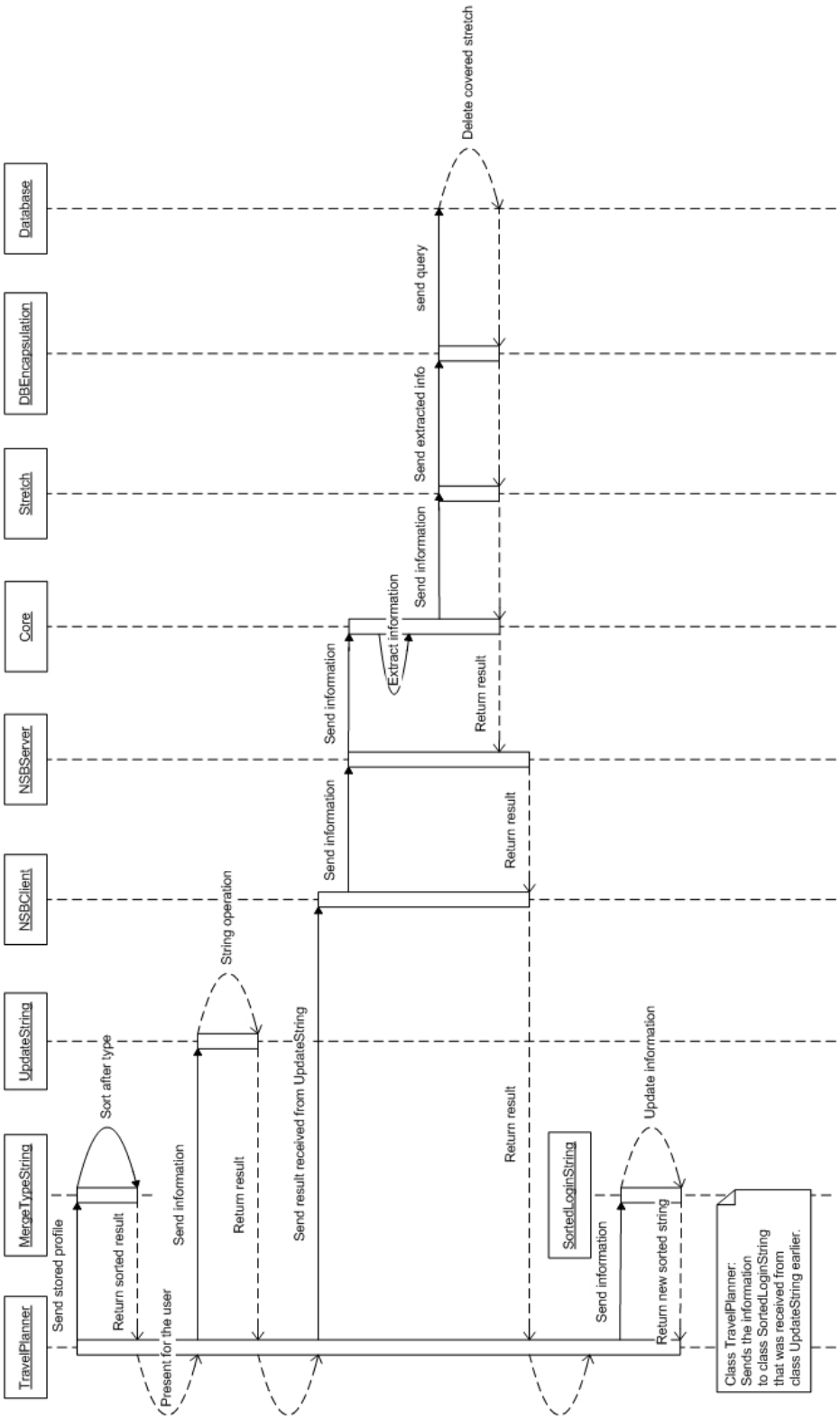

Figure 20: Sequence Diagram: Delete Route

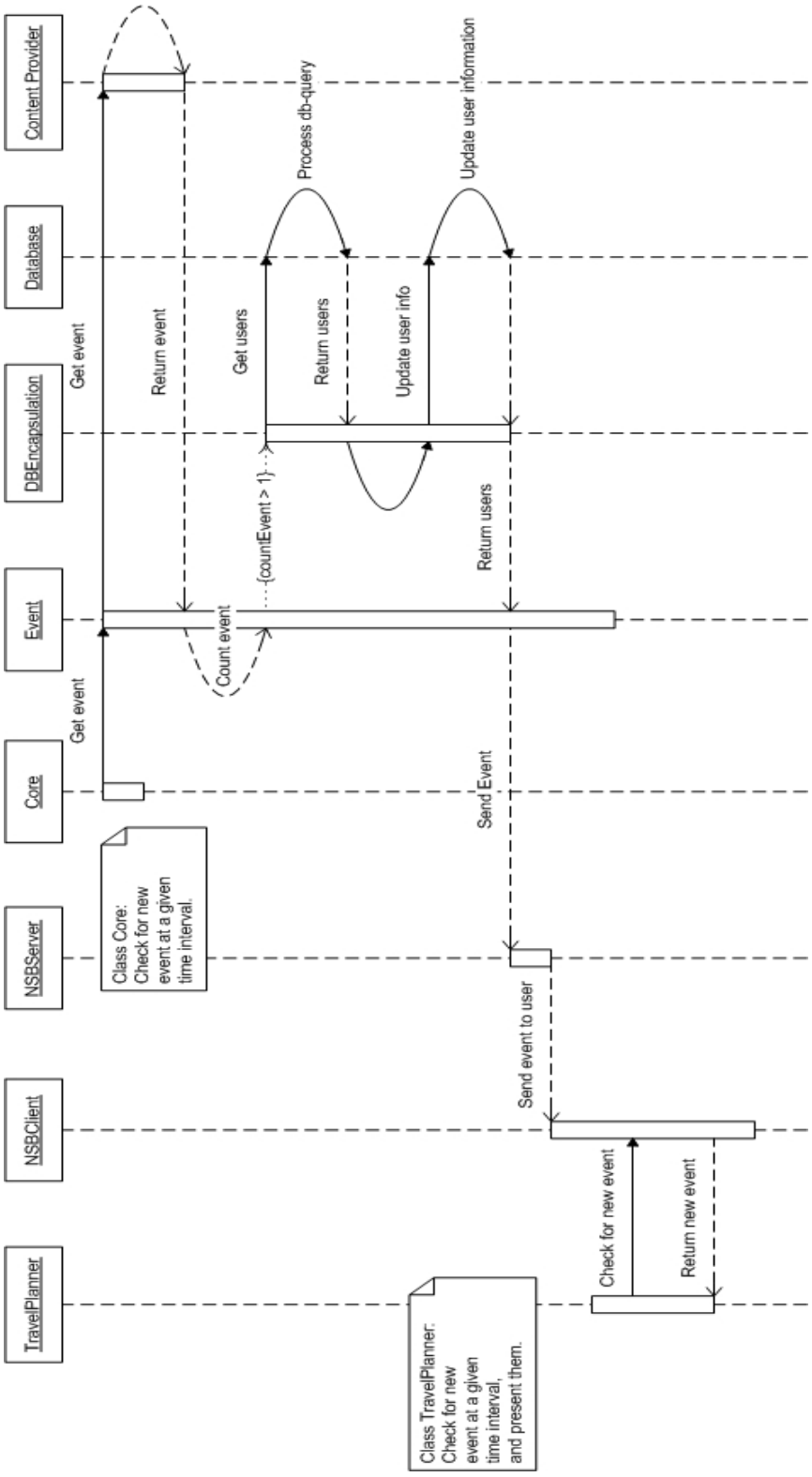

Figure 21: Sequence Diagram: Event Arrived

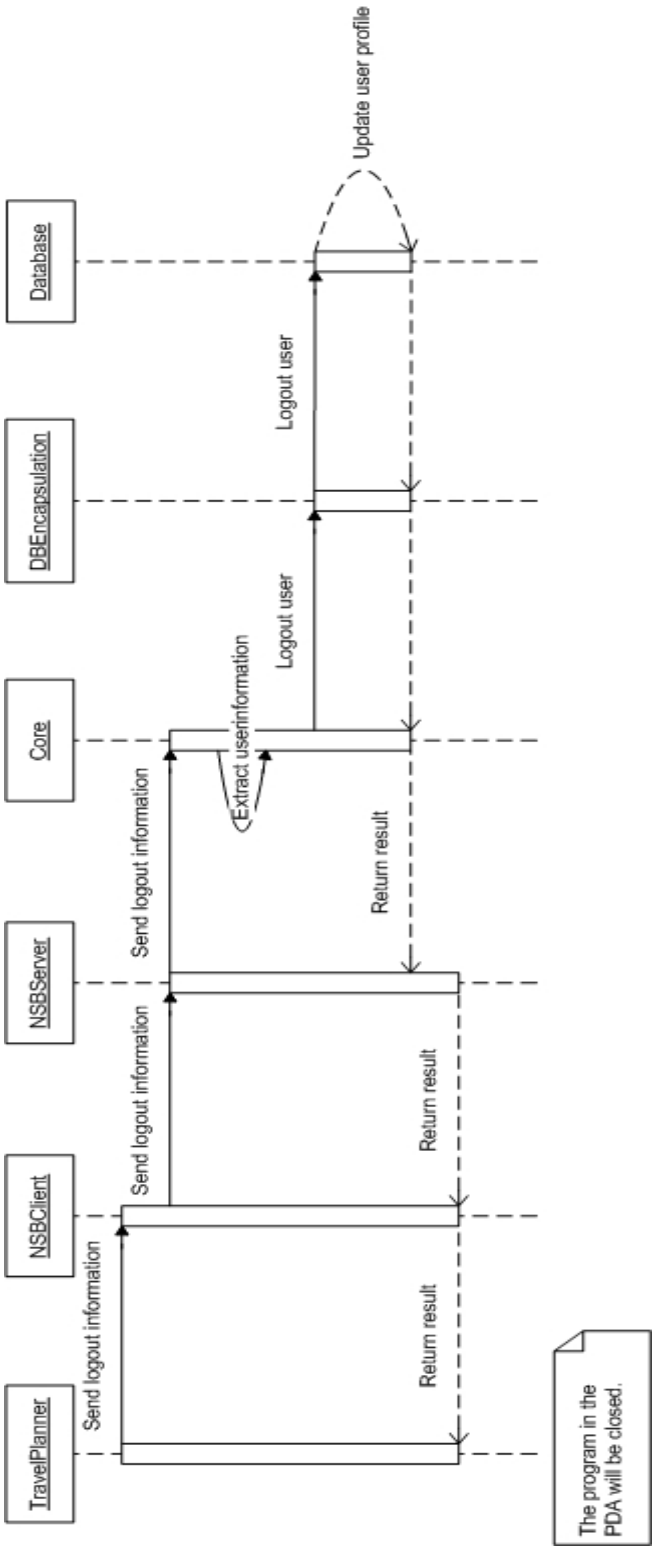

Figure 22: Sequence Diagram: Logout

Appendix D – User Manual

Table of Contents

1. Welcome Page ---------- **84**
1.1. Main Area (arrow 1) ---------- 84

2. The Menus ---------- **85**
2.1. Start Menu (arrow 2) ---------- 85
2.2. Routes Menu (arrow 3) ---------- 86
2.3. Help Menu (arrow 4) ---------- 88

3. The Various Pages ---------- **89**
3.1. Load My Profile Page ---------- 89
3.2. Create New Covered Stretch Page ---------- 91
3.3. Update My Covered Stretch ---------- 92
3.4. Search Route Page ---------- 93
3.5. Event happens ---------- 95

Table of Figures

Figure 24: Start menu **85**
Figure 25: Routes Menu **86**
Figure 26: Update My Covered Stretch **87**
Figure 27: Create New Covered Route **88**
Figure 28: Help Menu **88**
Figure 29: Load My Profile Page **89**
Figure 30: Create New Covered Route **91**
Figure 31: Update Covered Route **92**
Figure 32: Search Route **93**
Figure 33: My Route **94**
Figure 34: Event Happens **95**

The Applications Design

1. Welcome Page

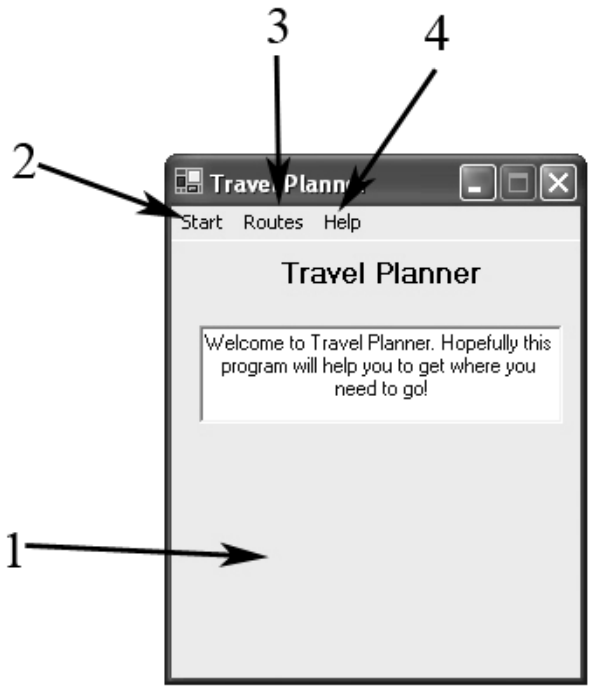

Figure 23: Welcome Screen

1.1. Main Area (arrow 1)

This is the welcome screen that the user will see when starting the application. It is not perfectly correct, since the only way to create a screenshot was to run the application on a PC instead on the PDA, which it is created for. The menu (arrows 2, 3, 4) is placed on the bottom of the screen when the application is run on a PDA.

Figure 24 shows the most important parts of the application. The most important of them all is the area which has an arrow with the number 1 attached to it. This area is the main area in which all information will be shown. Also here, all the options and possibilities for the user to maintain his application will be displayed. Things that could be presented in this area are text, textboxes, checkboxes buttons and scrollbars. Almost all these things are changeable by the user.

2. The Menus

There are three main menus in our application. These are:

"**Start**"
"**Routes**"
"**Help**"

All of these main menus contain some sub-choices and/or sub-menus. The description of them is seen below.

2.1. Start Menu (arrow 2)

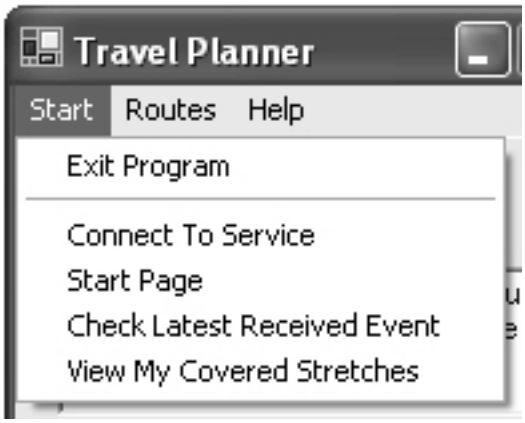

Figure 24: Start menu

The arrow with the number 2 attached to it in figure 24, points to the start menu. In this menu the most important features are available. According to figure 25, the start menu has five sub-choices. They are:

"**Exit Program**"
"**Connect to service**"
"**Start Page**"
"**Check Latest Received Event**"
"**View My Covered Stretches**"

Between the Exit and the Connect choice, there is a separator. This is placed there to really point out that Exit is a strong choice, it will terminate everything. All other choices in the menus will not kick out the user from the application.

Exit Choice
As said above, this choice will terminate the application. The user will not get a question to confirm the exit; the application will just shut down. After this is done, the application will not be able to receive any more events.

Connect To Service Choice
This is the “connect to service” choice. When the user has clicked on this choice, the user will be able to receive event, manage his covered routes and make new searches. The things that happened in the background are that the users profile is collected from the database on the server. This information is temporary stored on the PDA to make the use of the application faster and smoother.

Start Page Choice
This menu-choice will bring to front the start page.

Check Latest Event Choice
This choice will check for the latest received event, and display it on the Event page. By the “latest received event” we mean the last time that the user received an event (was displayed on the PDA). It is possible that more than one event is displayed while the user uses the application.

View My Covered Stretches Choice
When this choice is made, the profile will be brought to the front. All of the user routes that are stored in the database and still being valid are received and displayed on a text box with a scrollbars attached to them.

2.2. Routes Menu (arrow 3)

Figure 25: Routes Menu

As shown by figure 26, the search menu has four sub-choices, they are:

"**My Travel Route**"
"**Update My Covered Stretch**"
"**Create New Covered Stretch**"
"**Search For New Route**"

My Travel Route Choice
Here will the user's route be displayed. All information which concerns the user on his travel will be provided here, such as every substitution of busses, departure times and arrival times.

Update My Covered Stretch Choice
This is the first sub-menu under search route menu. It looks as shown in figure 27:

Figure 26: Update My Covered Stretch

Also this menu has four sub-choices, just like the "My Covered Stretch" menu. The choices are:

"**Bus**"
"**Subway**"
"**Train**"
"**Roads**"

When some of these choices are clicked it brings to front the update page for the different routes. If for example "Bus" is clicked on, then all covered routes that are for busses are selected from the database. Then the user could change the data that belongs to any bus in an easy way.

Create New Covered Stretch Choice
This menu-choice is a sub-menu to the search route menu. When this menu is opened it looks as shown in figure 28:

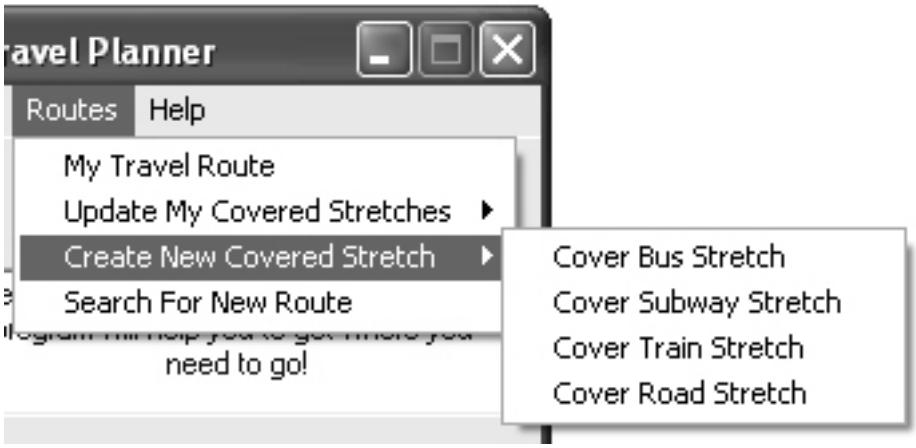

Figure 27: Create New Covered Route

This new menu that appears contains for choices. They are:

"**Bus Stretch**"
"**Subway Stretch**"
"**Train Stretch**"
"**Road Stretch**"

Then these choices are clicked on, a new page will be displayed. On this page, the user can insert new route to cover. The covered routes are divided into four sub-categories. They are Bus, Subway, Train and Road. What the user should chose depends on what kind of transportation that will be used.

Search For New Route Choice
This choice will bring up the page where the user makes his searches for new routes.

2.3. Help Menu (arrow 4)

Figure 28: Help Menu

This menu contains only two choices, they are:

"**Help Menu**"
"**About**"

Help Menu Choice
The help menu choice brings up the help page. On that page there will be displayed useful information about the application that will help the user to work with the application.

About Choice
This menu-choice will display information about, us who has done the application, and some information about the application itself.

3. The Various Pages

3.1. Load My Profile Page

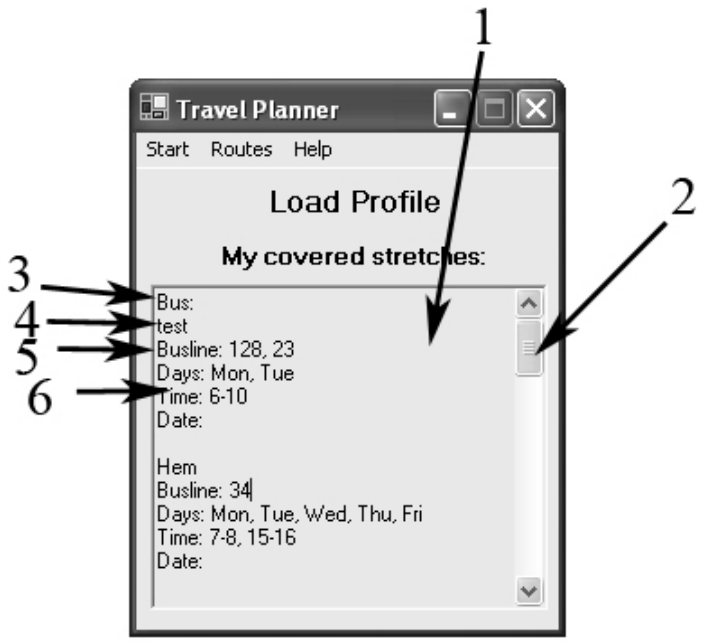

Figure 29: Load My Profile Page

Figure 30 shows the site which will appear when the user logs in and chooses to load his profile from the file menu. Here are six important things pointed out, these will be explained more closely.

Arrow 1 shows the area where the information about every covered route will be displayed. This area's size is dynamic, which means that it changes size according the amount of information that will be displayed there. This is possible thanks to the scrollbar on the right size (arrow 2). This scrollbars is attached to the textbox that contains the text.

Let us now have a closer look at the information in the textbox. First of all there is a word, Bus in this case, which denotes which type of transportation it addresses (arrow 3). Subsequently, all routes containing busses will be displayed. Under transportation type, the name of the route (arrow 4) is found. After that, the bus lines of interest for that specific post are specified (arrow 5). Finally, relevant date and time information is given. "Days" addresses the sorted weekdays and "Time" addresses the sorted hours (arrow 6). After time there could be another non-mandatory field, and it is the Date field. This will only be displayed if the user wants to cover a route during a specific date. After this there is a one row space before the next post is listed.

3.2. Create New Covered Stretch Page

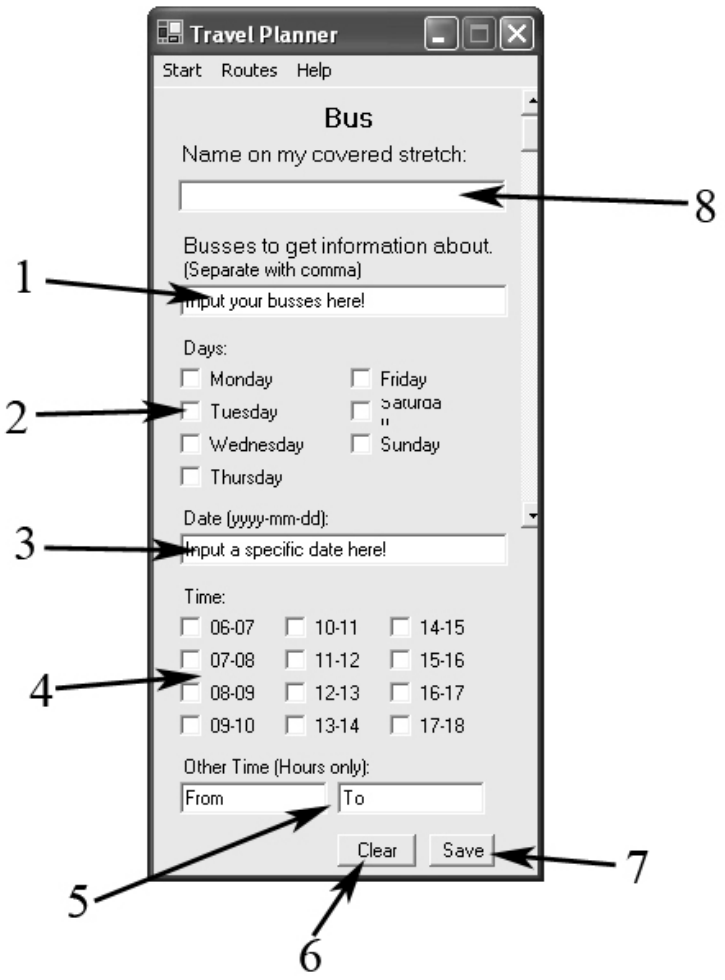

Figure 30: Create New Covered Route

The screenshot shown in figure 31 is stretched out so that the whole page can be shown. If it were displayed on the PDA, then the page would only be as long as the scrollbar to the right. This is what the user faces when creating a new covered route (in this case a bus stretch). Arrow 8 point to the textbox where the name of the specific route should be input, for example “Work”. Arrow 1 points on the textbox where the user will input the bus lines to be covered. For example, for bus line 4 and bus line 9, he will input 4, 9. Arrow 2 points to the checkboxes where the user will chose which days he wants to get information about. If the user wants to get information about a specific date, then he will input this date in the textbox that arrow 3 is pointing to. Arrow 4 points to the time checkboxes; here the user can decide which time he wants to get information about. The textboxes that arrow 5 is pointing at will contain information if the user wants to cover a route and doesn’t think that any of the checkboxes fits his or her needs (for example 19-20). Then he could use these textboxes to type in the interval. Only

whole hours are allowed here. If the user wants to reset the values he has, then he could click in the Clear button that arrow 6 is pointing to. When the user is finished with specifying the information about the route, then he clicks on the save button arrow 7 and the information will be stored into the database.

3.3. Update My Covered Stretch

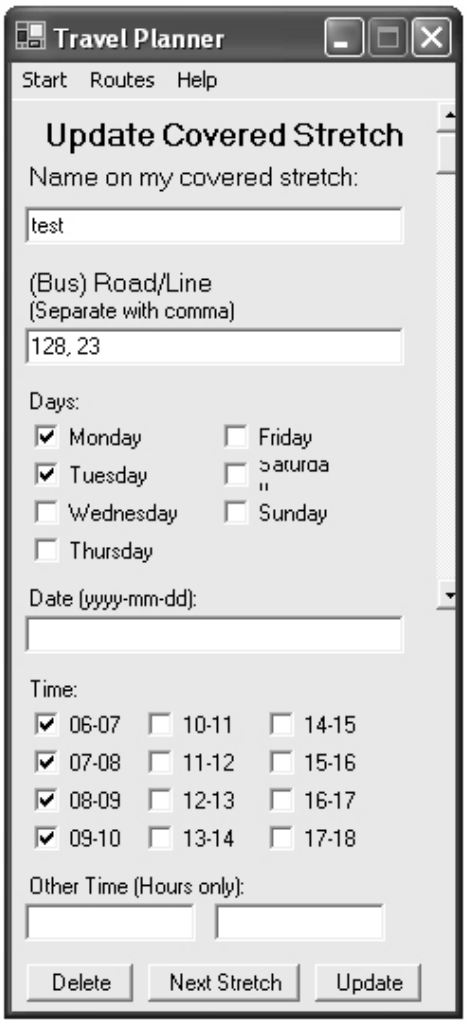

Figure 31: Update Covered Route

If the user chooses to update his covered route, he or she gets what is shown in figure 32. If it is a bus stretch that he wants to update, then he clicks on Bus in the menu. Then all bus stretches will be readily available so that he can step through them one in a time. This is possible thanks to the “Next Stretch” button.

When a route is loaded into the page, all of the information that belongs to that route is displayed. The user can see which line it addresses, and, which days and time the route covers.

If the user has stepped through all of his covered routes, then it will say “No more result for this type!” in the “Line” textbox on the top of the page. In this case, the Delete button won’t be visible.

3.4. Search Route Page

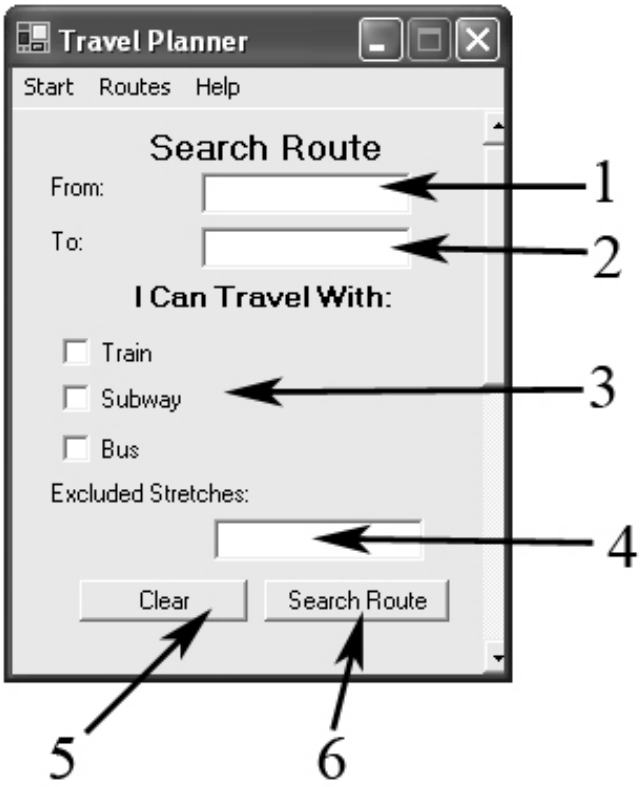

Figure 32: Search Route

On the search route page as shown in figure 33, the user will make his searches for new routes. The user inputs the name of the departure name (arrow 1) and the name of the arrival (arrow 2). Then the user specifies which kind of transportation types he can travel with (arrow 3) and which routes he wants to exclude (arrow 4). If he wants to reset all the information he clicks on the clear button (arrow 5). But if he thinks that the information is correct, then he clicks on the Search Route button (arrow 6). When the user clicks on the Search Route button, then the information will be forwarded to another page that will be brought to the front. This page is called My Route.

Travel Planner
Start
Routes
Help
My Route
Från Slussen onsdag 4 januari kl 15:33 till Globen kl 15:40
Ta tunnelbanans gröna linje från Slussen mot Hagsätra.
Den går från Slussen 15:33.
Du är framme vid Globen 15:40. Restid 7 minuter.
Trevlig resa!
Add to my profile
New Search

Figure 33: My Route

Here, the user can either make a new search or add the route to his profile as shown in figure 34. If the route is added to the profile, then the user will receive messages about events that concern the route.

3.5. Event happens

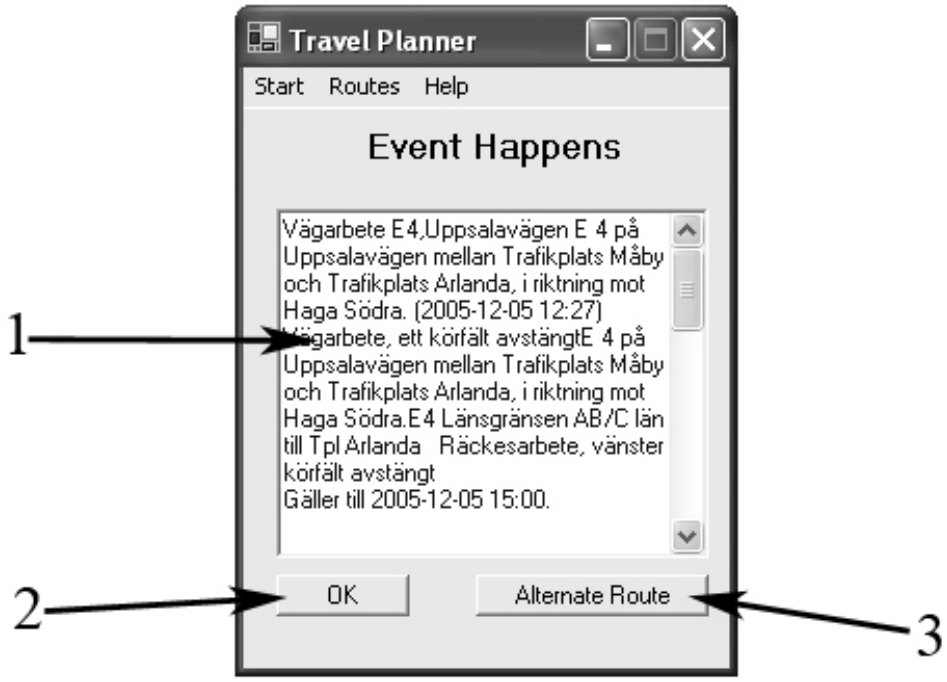
Travel Planner
Start Routes Help
Event Happens
Vägarbete E4,Uppsalavägen E 4 på Uppsalavägen mellan Trafikplats Måby och Trafikplats Arlanda, i riktning mot Haga Södra. (2005-12-05 12:27)
Vägarbete, ett körfält avstängtE 4 på Uppsalavägen mellan Trafikplats Måby och Trafikplats Arlanda, i riktning mot Haga Södra.E4 Länsgränsen AB/C län till Tpl Arlanda Räckesarbete, vänster körfält avstängt
Gäller till 2005-12-05 15:00.
OK
Alternate Route
1
2
3

Figure 34: Event Happens

Figure 35 shows how it looks when an event has arrived. To be able to receive events, the user has to be connected to the system (Start→Connect To Service). Then the system knows that the user is active.

The most important thing on this page is the event textbox in the middle (arrow 1). There, all events will be displayed. If several events arrive at the same time, they will be listed beneath each other. This is possible thanks to the scrollbar to the right. With it, the user can scroll down the textbox and view all events.

When an event has arrived, the user could either click on the OK button (arrow 2). This will tell the system "OK, I have seen the event. But I won't do anything about it!" Or the user could click on Alternate Route button (arrow 3). This will tell the system "OK, I have seen the event. I need a new route!"

If the user wants to receive any more events he has to click one of these buttons. Or else the events will be stored in the system until the user clicks on either of these buttons. In this way, no events will be lost.

Appendix E – Configuration Manual

Table of Contents

1. INSTALLATION INSTRUCTIONS ----------98

1.1. Database installation---------- 98

1.2. Server installation---------- 98

1.3. Client installation ---------- 98

2. APPLICATION CONFIGURATION ----------99

2.1. Server Side---------- 99
2.1.1. Database connection---------- 99
2.1.2. Port numbers ---------- 99
2.1.3. Addresses ---------- 99

2.2. Client side---------- 100
2.2.1. Address---------- 100
2.2.2. Port numbers ---------- 100

1. Installation instructions

To be able to install the software some prerequisites things need to be done. These prerequisites and how to get the application running are specified below.

1.1. Database installation

The application is using the MySQL database. This means that the MySQL Server must be installed on the computer that should act as the server in the application. Download MySQL Server 5.0 from www.mysql.com and follow the installation instructions provided by MySQL. When MySQL Server is up and running, create a new user account with username “piitsa” and password “piitsa”. After that, create a new database called “piitsaDB”. Set the user rights for the database so that the user can access the database. These actions can be performed in the “MySQL Command Line Client”, which is included in the MySQL server package, or by using “MySQL Administrator”. To get the tables into the database import the “piitsa.sql” file. This could be done by either using the “MySQL Command Line Client” or by using “MySQL Query Browser”. Both the administrator tool and the query browser are available for download at www.mysql.com.

1.2. Server installation

To be able to run the application on the server side the database must be working, see above. To get the application working with the privious installed database, the address to the database must be configured; see section 2.1.1 in this appendix on how to configure the database connection. The application is an executable file with the file extension .exe. The file ServerCore.exe is located in the directory “ServerCore\bin\debug”. If a firewall is used, make sure that it allows connections to the program. Another important thing for the server side is that the RSS-file, RSS.xml is located in the same directory as ServerCore.exe.

1.3. Client installation

To transfer the application to the PDA there are a few step to perform. First, make sure that the application on the PDA isn’t running. Connect the PDA, without the GPRS jacket, to the PC through the docking station. Start the Visual studio project “Travelplanner.csdproj”. Configure the server address. To do this, see section 2.2.1 in this appendix. Re-build the application by selecting “Pocket PC Device” in the drop down menu in the toolbar and pressing “f5” will transfer the application to the PDA. The application will be located in “Progarm\TravelPlanner” on the PDA. Close the application and disconnect the PDA from the PC. Connect it to the GPRS jacket and get a GPRS connection. At last, start the application again.

2. Application configuration

There are a few parameters that are possible to change, both on the server side and on the client side. These parameters and their location ar specified in the following.

2.1. Server Side

On the server side, there are mainly four things that could be configured. It is the connection to the database, which port number should be used for communications between the client and the server, how often the server should check for new events, and the addresses used when checking events or when searching routes. All configurations are done in Visual Studio. After the change has been done, re-compile the project by pressing “f5” and a new version of the application is generated and stored in the directory “ServerCore\bin\debug”

2.1.1. Database connection

The configuration of the database connection is made in the file called DBEncapsulation.cs. At the beginning of the file, in the constructor the connection string is located. The information in this string is depending on what kind of database is being used. The easiest way to get the right information is to visit the database supplier’s website, or www.connectionstrings.com [10] providing connection information about the most popular databases.

2.1.2. Port numbers

Another vital thing that could be configured is the port numbers the server use to listen for connections from the client. In the application we use two ports for the communication, one for event communication and one for all the other communication. These port numbers must be the same on both server side and client side, see section 2.2 in this appendix.

On the server side, the port numbers are configured in the file called NSBCore.cs. They are located on the first two rows in class NSBCore. TCPPORT is used for all communication except for event and TCPEVENTPORT is used for all the event communication. If the server is running behind a firewall or router, the firewall or router also needs to be configured so it allows connections to the specified ports in NSBCore.

2.1.3. Addresses

There are two different kinds of addresses that could be configured on the server side. The first one is where the server gets information about event. This is configured in the file called Event.cs. The parameter is set in RSS, located on the

first row in class Event. The value can be either a local file or an http address. The program is designed for reading an rss/xml file from trafiken.nu. If the RSS file isn't built up in the same structure as provided by trafiken.nu it may not work at all.

The other address that could be configured is the query address to SL. This is harder to configure, since it's a third party where the information is gathered. If SL changes a name on any of their variables that we send to them, the search function doesn't work until our address is re-configured. To re-configure the address you have to view the source code generated on their website and get all necessary information about the variables. This is time-consuming, but it has to be done to get it working again.

2.2. Client side

On the client side, there are two parameters that can be configured. It is the address to the server and which port it communicates on. These parameters is also configured in Visual Studio, which means that the project must be re-compiled for the changes to take place. Make the changes and re-compile and transfer the application to the PDA. See 1.3 in this appendix how to transfer the application to the PDA.

2.2.1. Address

There is one address that could be re-configured. It is the address to the server side. This parameter, ADDRESS, is located on one of the first rows in class NSBClient in the file NSBClient.cs.

2.2.2. Port numbers

There are two port numbers that could be configured, the port number for all communication to the server except event communication and the number of the port that handles all event communications to the server. These parameters, PORT for all communications except event and EVENTPORT for event communications, are located on one of the first rows in class NSBClient in file NSBClient.cs. It is important that these port numbers are configured exactly as the port numbers on the server side, otherwise there won't be any connection between the client and the server.